Escuela Técnica Superior de Ingeniería
Universidad de Huelva

Grado en Ingeniería Informática

Trabajo Fin de Grado

Clasificación de imágenes de fototrampeo con una red neuronal convolucional

Autora:
Alba Márquez-Rodríguez

Tutores:
Manuel Emilio Gegúndez Arias,
Diego Marín Santos

Huelva, Octubre 2023

Resumen

El estudio de la biodiversidad conlleva la observación directa en el campo o análisis de imágenes, suponiendo una carga en tiempo y esfuerzo para especialistas. En este contexto, este Trabajo Fin de Grado aborda el diseño y aplicación de un modelo de clasificación diseñado para imágenes de fototrampeo.

Para su desarrollo, se ha utilizado un conjunto de 8000 imágenes de fototrampeo clasificadas según la especie que contiene: caballo, ciervo, gamo, humano, jabalí, vaca, vacía y zorro. Como modelos de clasificación, se han utilizado dos redes neuronales convolucionales con dos niveles de profundidad, *MobileNetV2* (3.088.680 parámetros) y otra red llamada *Arquitectura de Crohn* (599.168 parámetros). El entrenamiento, validación y evaluación de estas redes se llevó a cabo con una distribución del 80 %, 10 % y 10 % del conjunto respectivamente. La tasa de acierto de las redes se situó entorno al 70 %. Las distintas pruebas realizadas para analizar los resultados mostraron que la principal causa de este bajo rendimiento se situaba en la dificultad inherente que presenta la extracción de conocimiento en las imágenes tratadas, unido al número reducido de imágenes disponibles. Esta conclusión se alcanzó tras aplicar las redes a una nueva base de datos de 21000 imágenes y 15 clases diferentes donde los elementos de interés se muestran en primer plano y en un entorno más acotado. La tasa de acierto de las redes sobre esta base de datos de referencia fue superior al 95 %.

En resumen, este estudio subraya la importancia de contar con un conjunto de datos de alta calidad, ya que esto tiene un impacto directo y significativo en el rendimiento de las redes neuronales. Los resultados que se pueden lograr con una red están intrínsecamente vinculados a la calidad del conjunto de datos y las anotaciones disponibles.

Palabras clave: aprendizaje profundo, aprendizaje automático, inteligencia artificial, procesamiento de imágenes, redes neuronales convolucionales, conjunto de datos.

Abstract

Biodiversity studies involve direct field observations or the analysis of images, which presents a workload in terms of time and effort for specialists. In this context, this Bachelor's Thesis addresses the design and application of a classification model for camera trap images.

An 8000-image dataset from camera traps have been employed, classified by contained species, including horse, deer, fallow deer, human, wild boar, cow, and fox. Two convolutional neural networks were used as classification models: *MobileNetV2* (3,088,680 parameters) and a network named *Crohn's Architecture* (599,168 parameters). Training, validation, and evaluation were carried out with an 80 %, 10 %, and 10 % split, respectively. The networks achieved an accuracy rate of approximately 70 %. In-depth analysis unveiled the primary challenge of knowledge extraction from these images due to their intrinsic complexity, amplified by the limited dataset size. This observation materialized upon applying the models to a novel dataset containing 21,000 images spanning 15 diverse categories. These images showcased the subject of interest prominently against a controlled backdrop. The networks' performance on this reference dataset significantly improved, yielding an accuracy rate exceeding 95 %.

In summary, this study emphasizes the importance of having a high-quality dataset, as it directly and significantly impacts the performance of neural networks. The achievable results with a neural network are inherently linked to the dataset's quality and available annotations.

Key words: deep learning, machine learning, artificial intelligence, images processing, convolutional neural networks, dataset.

Índice

Índice de figuras

1. Propuesta de Proyecto

A continuación se hace una presentación del proyecto, recogiendo la motivación detrás de este, los objetivos que lo impulsan y cómo se llevará a cabo.

1.1. Motivación

Actualmente se vive el auge del campo de la informática, más concretamente, del de la inteligencia artificial. Mientras a lo largo de la historia, los seres humanos han tratado de entender *cómo piensan*, el campo de la inteligencia artificial no sólo intenta comprender, sino que también intenta construir entidades inteligentes [1].

Dentro de la inteligencia artificial hay varios campos tal y como se muestra en la Figura 1, uno de los que más repercusión tiene hoy en día y en el que se busca y espera avanzar más en los próximos años es el de visión por computador, visión artificial o *computer vision*.

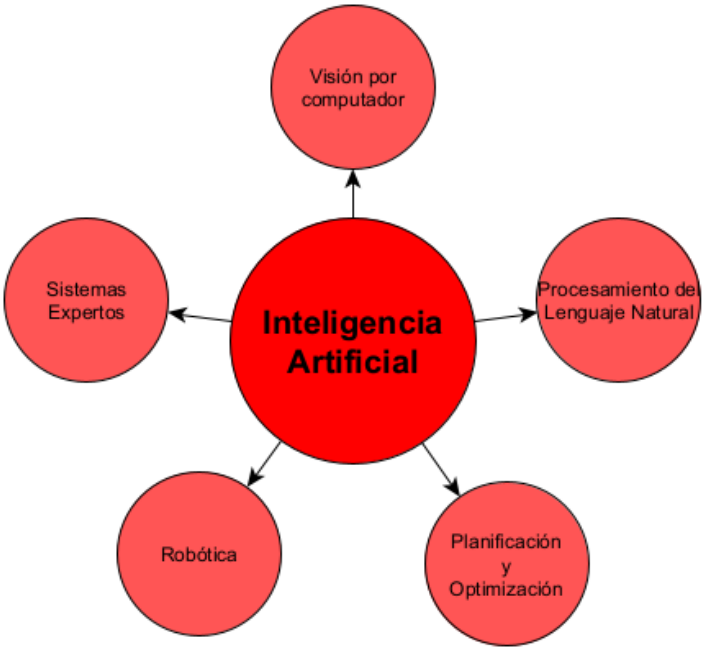

Figura 1: Campos de la inteligencia artificial

Fuente: Elaboración Propia

En el campo de la visión artificial, las máquinas se entrenan para extraer y comprender el contenido de una imagen [2]. Este avance se ha vuelto posible gracias al aprendizaje profundo, también conocido como *deep learning*, que permite que los modelos computacionales aprendan de un conjunto de datos de entrenamiento, compuesto por imágenes clasificadas y etiquetadas manualmente, y realicen predicciones en un conjunto de datos nuevo, que contiene imágenes sin etiquetar [3]. Este enfoque se basa principalmente en la técnica de inteligencia artificial de redes neuronales.

La visión por computador, utilizada para el análisis de imágenes, ha demostrado su capacidad para resolver problemas como la clasificación, detección y segmentación, ejemplificados en la Figura 2, y se aplica en campos tan diversos como la medicina, la robótica y la ecología.

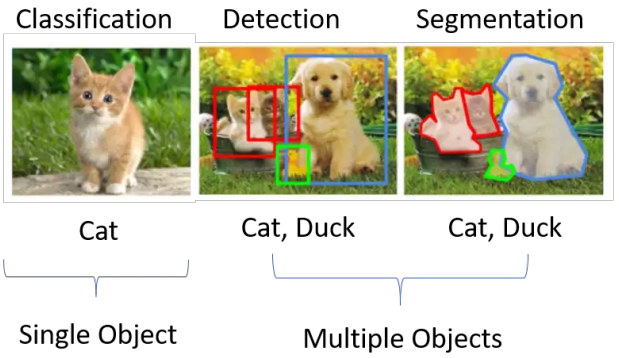

Figura 2: Ejemplo de clasificación, detección y segmentación

Fuente: https://medium.com/@kolungade.s/object-detection-image-classification-and-semantic-segmentation-using-aws-sagemaker-e1f768c8f57d

Dentro del campo de la ecología, ha surgido un interés creciente en el uso del *deep learning* para automatizar el análisis repetitivo de grandes cantidades de imágenes. Esto abarca desde la identificación de especies hasta la distinción del número de individuos de diferentes especies, su conteo y la detección de características relevantes [4].

Este Trabajo Fin de Grado (TFG) tiene como objetivo adentrarse en el campo de la inteligencia artificial y su aplicación en la ecología. La motivación central de este proyecto es abordar el desafío del análisis de imágenes de fototrampeo para lograr la clasificación de diferentes especies.

1.2. Objetivos

Mediante la realización de este trabajo se plantea cumplir una serie de objetivos que, a continuación, se detallan.

1.2.1. Objetivos de Formación

- Introducción al *deep learning*: Se realizará una introducción al campo del *deep learning*, abordando su definición, su desarrollo histórico y su estado actual.
- Estudio teórico del *deep learning* y Redes Neuronales: Se llevará a cabo un estudio teórico de los fundamentos del *deep learning*, profundizando en los conceptos y técnicas fundamentales utilizadas en este campo. Se explorarán los fundamentos y conceptos clave del *deep learning*, incluyendo arquitecturas de redes neuronales profundas, algoritmos de aprendizaje, aplicaciones relevantes en diversos dominios y las diferentes etapas de las que se puede componer. Este estudio teórico proporcionará una base sólida para comprender el funcionamiento interno de las redes neuronales profundas y su capacidad para aprender de manera automática a partir de los datos.
- Estudio teórico de las Redes Neuronales Convolucionales: al ser el objetivo implementar y aplicar una red neuronal de clasificación de imágenes, se profundizará en las redes neuronales convolucionales. Se examinarán en detalle los diferentes tipos de capas presentes en las redes neuronales convolucionales, así como las arquitecturas y modelos utilizados en este contexto. Además, se estudiarán los algoritmos de entrenamiento y las estrategias de evaluación para mejorar el rendimiento del modelo.
- Estudio de la influencia del *dataset*: se analizará la influencia del *dataset* en los resultados de la red neuronal. Se utilizará un *dataset* de clasificación de verduras obtenido de Kaggle, que ha sido utilizado en concursos para lograr el mejor accuracy posible. El objetivo es comparar el rendimiento de diferentes redes neuronales en este *dataset* y demostrar cómo los resultados pueden variar dependiendo del *dataset* utilizado.

1.2.2. Objetivos de Aplicación al caso Práctico

- Implementación práctica de algoritmos de *deep learning* para la clasificación de imágenes.
- Obtención y preparación de los datos necesarios para los algoritmos a utilizar.
- Diseño y evaluación del sistema de clasificación de imágenes del *dataset* de animales (dataset de un problema real de clasificación).
- Diseño y evaluación del sistema de clasificación de imágenes del *dataset* de verduras (dataset preparado para el estudio de modelos de *deep learning*).
- Evaluación de la influencia de los diferentes *dataset*s sobre los mismos modelos de redes neuronales.

Con estos objetivos, el trabajo se centrará en intentar clasificar especies de animales en imágenes de fototrampeo con Redes Neuronales Convolucionales. Los resultados esperados no son buenos por los que se demostrará que los malos resultados se deben al *dataset* utilizado, un *dataset* sumamente complejo y con información insuficiente para abordar el problema.

1.3. Hardware y Software

1.3.1. Hardware

Para poder conseguir los objetivos descritos, es necesario disponer de un equipo capaz de satisfascer las necesidades de los algoritmos, ya que una de las mayores exigencias de estos es la potencia del hardware al requerir de una gran capacidad de cálculo, que, además puede ser realizado de forma paralela.

Para la implementación de los modelos y el desarrollo de los modelos se usará el ordenador portátil de la alumna. Mientras que para el entrenamiento y la experimentación de las redes neuronales finales se realizará en el ordenador de — debido a la alta capacidad de cómputo que requiere una red para el entrenamiento.

1.3.2. Software

Con respecto al software, la implementación de las redes neuronales y diferentes transformaciones se harán en el lenguaje de Programación *Python 3.10*. Este lenguaje ha sido escogido por la gran popularidad de uso en aplicaciones de inteligencia artificial. Existen muchos *frameworks* que contienen funcionalidades necesarias para la implementación de modelos de *deep learning*.

Figura 3: *Frameworks* para *deep learning*

Fuente: https://medium.com/apache-mxnet/a-way-to-benchmark-your-deep-learning-framework-on-premise-4f7a0f475726

La elección del *framework* se puede basar en la calidad de la documentación y el respaldo por parte de la comunidad de usuarios que se dedican al desarrollo del *deep learning*.

El framework elegido es Tensorflow 2.0, que cuenta con una documentación detallada y una comunidad muy activa. Además incluye de forma nativa desde el framework Keras, que permite la creación de redes neuronales desde un alto nivel gracias a su estructura de capas de abstracción.

Para la implementación y el entrenamiento de estos modelos se hará uso de Pycharm y Jupyter Notebook, que junto con herramientas de Python nos permitirán crear un entorno de desarrollo dónde poder instalar las librerías necesarias para el desarrollo del trabajo tales como TensorFlow, OpenCV, Numpy y otras.

1.4. Organización del documento

Además de este primer capítulo, en el cual se ha presentado la propuesta del proyecto, esta memoria se ha estructurado en los siguientes capítulos:

- **Capítulo 2, Introducción al *Deep Learning***: En este capítulo se introduce el *deep learning*, sus antecedentes y el estado del arte.
- **Capítulo 3, Redes Neuronales**: En este capítulo se realiza una introducción teórica a las técnicas de *deep learning*. La evolución histórica, arquitecturas generales y las etapas de las que se compone un algoritmo basado en redes neuronales.
- **Capítulo 4, Redes Neuronales Convolucionales**: Se profundiza en las técnicas de *deep learning* que vamos a usar en este proyecto, las Redes Neuronales Convolucionales. Explicando los diferentes tipos de capas y arquitecturas que ya se han utilizado.
- **Capítulo 5, Introducción al problema. Estado del arte**: En este capítulo, se aborda el problema propuesto de clasificación de especies de animales en imágenes de fototrampeo mediante el uso de CNN. Se explicará la motivación detrás de este problema y se realizará un breve estado del arte para comprender cómo se ha abordado anteriormente.
- **Capítulo 6, Materiales**: En esta sección, se presentan los materiales que fueron empleados en el desarrollo de este estudio. Los materiales incluyen los recursos y conjuntos de datos específicos que desempeñaron un papel crucial en la experimentación y evaluación de los modelos. Además, se destacará la relevancia de cada uno de estos elementos en relación con los objetivos y resultados del estudio.
- **Capítulo 7, Metodología**: A continuación se va a presentar la metodología utilizada para los diferentes modelos. Se abordarán las arquitecturas de redes neuronales convolucionales utilizadas en este proyecto, los hiperparámetros utilizados, las técnicas de *data augmentation*, el proceso de entrenamiento de los modelos, la optimización de parámetros y, finalmente, se presentarán las conclusiones y la selección de los modelos.
- **Capítulo 8, Evaluación**: En esta sección, se examinarán los resultados obtenidos por los modelos previamente seleccionados, con un enfoque especial en la evaluación de su rendimiento en un conjunto de datos completamente desconocido para ellos: el conjunto de *test*.
- **Capítulo 9, Aplicación a un *dataset* de entorno controlado**: Este capítulo se centra en la aplicación de los modelos desarrollados a un *dataset* de entorno controlado. Se analizará cómo los modelos entrenados en el problema de clasificación de especies de animales en imágenes de fototrampeo se desempeñan en un contexto diferente y se discutirán los resultados y las implicaciones de esta aplicación.
- **Capítulo 10, Conclusiones**: En este último capítulo, se presentarán las conclusiones generales del estudio. Se resumirán los hallazgos clave, se discutirán las limitaciones y se ofrecerán recomendaciones para futuras investigaciones en este campo de estudio.

2. Introducción al *Deep Learning*

Antes de comenzar a abordar la creación de sistemas basados en *deep learning*, en este capítulo se ofrece, en su primer punto, una introducción al *deep learning* que permite encuadrar esta disciplina como un campo específico del *machine learning*.

A contiuación, en el segundo punto de este capítulo se realiza un recorrido histórico.

Y, finalmente, en el tercer punto del capítulo, se realiza un estudio del estado del arte de esta tecnología, así como a algunas de sus aplicaciones.

2.1. Contexto General

Un ordenador funciona con *algoritmos*, una serie de pasos ordenados que debe realizar para completar una tarea determinada. Debido a ello, para que un ordenador pueda resolver un problema, debemos aportarle un algoritmo que lo solucione. Pero encontrar este algoritmo no es trivial. Por este motivo surge el aprendizaje automático o *machine learning*, como propuesta para encontrar un algoritmo capaz de resolver un problema.

La inteligencia artificial pretende que la máquina procese la información y la factorice en decisiones basadas en lograr algún resultado esperado.

El aprendizaje automático o *machine learning* es un área, dentro de la Inteligencia Artificial (como se muestra en la Figura 4), que consiste en crear el algoritmo de manera que este sea capaz de reconocer patrones en los datos sin que la persona que lo programe necesite especificar cómo el algoritmo debe llevar a cabo todos los aspectos de este reconocimiento [5]. Así se pretende que sea el propio sistema el que encuentre el algoritmo para resolver el problema.

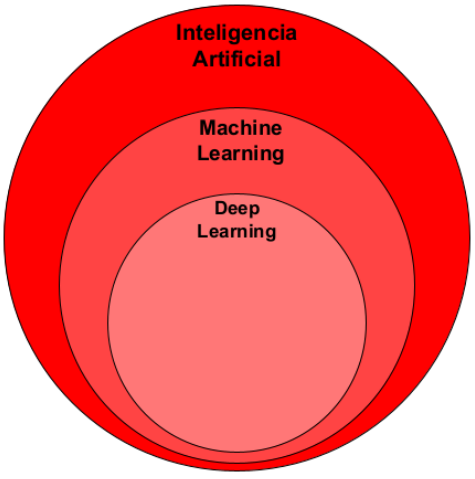

Figura 4: Posicionamiento de los conceptos de Inteligencia Artificial

Fuente: Elaboración Propia

De este modo el *machine learning* requiere de varias etapas. En la Figura 5 se muestra el proceso, pero de forma general se pueden distinguir dos:

1. **Entrenamiento**: Genera un modelo a partir de unos datos de entrenamiento en el que se estudian los patrones de los datos.
2. **Inferencia**: A partir del modelo generador en la etapa anterior, se somete a nuevos datos y se obtienen resultados.

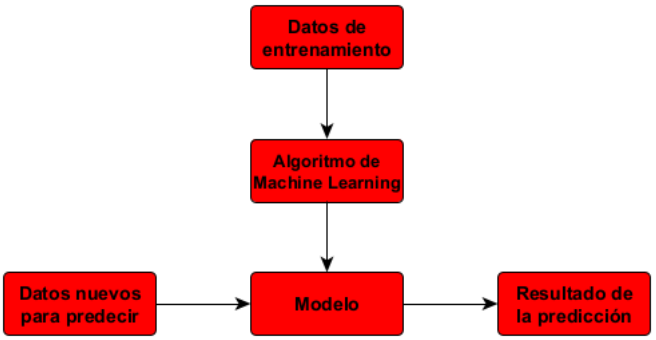

Figura 5: Proceso de *machine learning*

Fuente: Elaboración Propia

El *machine learning* engloba una gran variedad de algoritmos con diferentes propósitos. Según cómo se aborde el problema, existen diferentes tipos de aprendizaje, ejemplificados en la Figura 6:

- **Aprendizaje supervisado**: En este enfoque, se cuenta con información previa que asocia ciertos datos con resultados conocidos. El objetivo es encontrar una función que pueda predecir o aproximar estos resultados a partir de los datos disponibles.
  - **Clasificación**: Se aplica cuando los resultados a predecir son categorías o etiquetas específicas, es decir, variables categóricas.
  - **Regresión**: Se utiliza cuando los resultados a predecir son valores numéricos continuos.
- **Aprendizaje no supervisado**: En este caso, no se dispone de información que vincule los datos con resultados conocidos. El objetivo principal es descubrir patrones o estructuras intrínsecas en los datos.
  - ***Clustering*** o agrupamiento: Se emplea para dividir las entradas en grupos que comparten características comunes.
- **Aprendizaje por refuerzo**: Este enfoque implica un agente que interactúa con un entorno. Las acciones del agente tienen impacto en la información proporcionada por el entorno al agente, lo que permite al agente aprender a través de la retroalimentación de sus propias acciones.

Además, también podemos distinguir diferentes algoritmos, entre los más conocidos se encuentran:

- **Regresión Lineal *(Linear Regression)***: Es un algoritmo utilizado para la regresión, donde se busca encontrar la mejor relación lineal entre una o varias variables independientes y una variable dependiente. El modelo de regresión lineal busca minimizar la diferencia entre los valores predichos y los valores reales a través de una línea recta [6].

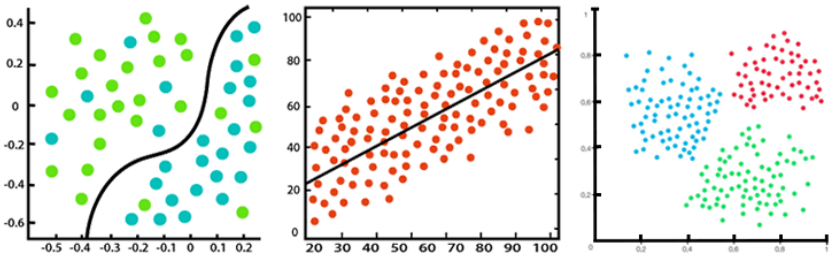

Figura 6: Ejemplo de, de izquierda a derecha, clasificación, regresión y *clustering*

Fuente: Elaboración Propia

- **Árboles de Decisión *(Decision trees)***: Puede ser utilizado tanto para la regresión como para la clasificación. Se genera un modelo representado con la estructura de un árbol. Comienza por la raíz y se mueve hacía los nodos hasta que una predicción puede ser hecha [7].
- **KNN *(K-nearest neighbors)***: Puede ser utilizado tanto para la regresión como para la clasificación. Consulta los k vecinos más cercanos, es decir, los k patrones de entrenamiento más similares [8].
- **SVM *(Support Vector Machines)***: Utilizado para la clasificación. Genera un modelo buscando la línea que mejor separa los datos de entrenamiento en diferentes clases [9].
- **Redes Neuronales *(Neural Networks)***: Las redes neuronales pueden ser utilizadas tanto para clasificación como para regresión. El modelo es generado a partir de un grafo formado por el modelo matemático de las neuronas biológicas. Las redes neuronales tienen la capacidad de aprender y reconocer patrones complejos en los datos, lo que las hace adecuadas para una amplia gama de problemas [10].

Dentro del campo del aprendizaje automático (*machine learning*), se encuentra una rama conocida como redes neuronales, que se basa en el estudio y desarrollo de algoritmos capaces de aprender y encontrar patrones en los datos. Estos algoritmos se inspiran en las neuronas biológicas y se organizan en capas, formando lo que se conoce como redes neuronales artificiales [11].

Una red neuronal típica consta de tres tipos de capas:

1. **Primera capa**: Capa de entrada, recibe los datos iniciales y los transmite a las capas ocultas para su procesamiento.
2. **Capas ocultas** o ***hidden layers***: Multiples capas que realizan operaciones matemáticas sobre los datos de entrada, extrayendo características y generando representaciones más abstractas.
3. **Última capa**: Capa de salida que devuelve los valores que la red produce como resultado final.

El *deep learning* se refiere al uso de redes neuronales profundas, es decir, redes con múltiples capas ocultas, para resolver problemas complejos. El perceptrón multicapa es considerado el modelo básico de *deep learning*, ya que con tan solo dos capas se puede obtener un aproximador universal de funciones [12]. Aunque en teoría es posible aproximar cualquier función con una sola capa oculta, en la práctica se requieren un número excesivamente elevado de neuronas, lo cual puede

ser computacionalmente costoso. Por ello, se prefiere aumentar la profundidad de la red, añadiendo más capas ocultas, lo cual permite resolver problemas con un menor número total de neuronas.

Estos algoritmos basados en *deep learning* se caracterizan por, al necesitar muchos parámetros para ajustar, necesitar una gran cantidad de datos de entrenamiento y un sistema con gran capacidad de procesamiento. Estos problemas han dado lugar al desarrollo de otras disciplinas como el *Big Data* para resolver los problemas de almacenamiento de gran cantidad de datos. Y, por otro lado, el desarrollo de *GPUs* con mayor capacidad de cómputo. Además se fundamentan en, como se mencionó anteriormente, las redes neuronales. Así que para hablar de los algoritmos también se debe hablar de la arquitectura de estas. En la sección 3.1.3 se mencionan y explican algunas de las arquitecturas más populares.

Durante los primeros años de investigación en redes neuronales, existía un gran interés en utilizar redes neuronales muy profundas para incrementar la capacidad de las redes. Sin embargo, el entrenamiento de redes neuronales profundas era un desafío debido a las limitaciones de capacidad de procesamiento y algoritmos de entrenamiento. No fue hasta 2006 cuando Geoffrey E. Hinton, Simon Osindero y Yee-Whye Teh lograron entrenar redes con un número mucho mayor de capas que las entrenadas previamente, lo cual impulsó a otros investigadores a adentrarse en el entrenamiento de redes neuronales más profundas. Este hito en el campo de las redes neuronales profundas condujo al término *deep learning* para referirse específicamente al entrenamiento de redes más profundas que las entrenadas hasta ese momento [13].

En resumen, el *deep learning* aprovecha las redes neuronales profundas para aprender y descubrir patrones complejos en los datos. Mediante capas ocultas adicionales, las redes neuronales profundas pueden representar características y relaciones más abstractas, permitiendo resolver problemas más complejos con un menor número de neuronas. El desarrollo de algoritmos de entrenamiento más eficientes y el aumento en la capacidad de procesamiento han impulsado el avance y aplicación del *deep learning* en diversos campos.

La Figura 7 muestra un ejemplo de estructura básica de una red neuronal.

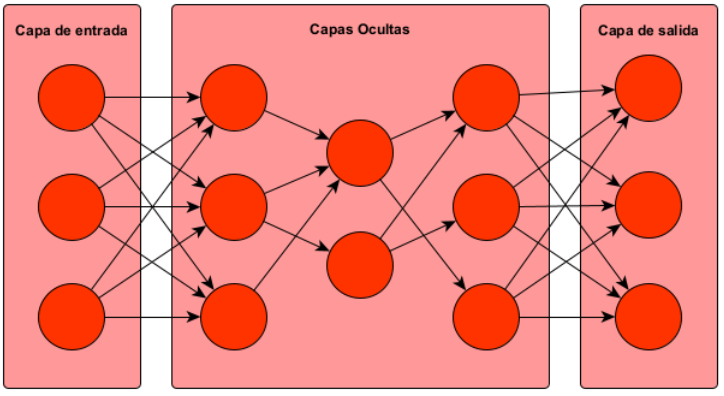

Figura 7: Ejemplo de una posible estructura básica de una red neuronal

Fuente: Elaboración Propia

A continuación se repasará la historia del *deep learning* a través del tiempo y los avances que han llevado al estado actual de este.

2.2. Antecedentes

Como se explicó en la sección anterior, para hablar del *deep learning*, primero hay que hablar del *machine learning* y, por supuesto, de la inteligencia artificial.

El estudio del *deep learning* y las redes neuronales tiene su origen en la ambición del ser humano de construir un sistema informático que simule el cerebro humano. Para construir un sistema de este tipo se requiere la comprensión de la funcionalidad de nuestro sistema cognitivo.

Aristóteles fue el inicio del *Asociacionismo*, que es una teoría que establece que la mente es un conjunto de elementos que se organizan como asociaciones entre estos elementos. Inspirado en Platón, Aristóteles examinó los procesos de memoria y el recuerdo y creó las leyes de asociación [14].

En aquellos tiempos Aristóteles las denominó como sentido común. Hoy en día aún sirven como las suposiciones fundamentales de los métodos de *machine learning*. Esta teoría del *Asociacionismo* fue posteriormente apoyada y reforzada por otras personas del ámbito filosófico como Tommas Hobbes, John Locke o David Hartley, quien hizo el *Asociacionismo* popular [15].

En el año 1936 Alan Turing publica un estudio [16] donde describe las *máquinas de Turing*, que formalizan el concepto de algoritmo, y que es considerado como el comienzo de la computación moderna.

En 1943 Warren McCulloch y Walter Pitts presentan un modelo matemático de las neuronas biológicas, considerado como el primer trabajo en el campo de la inteligencia artificial, aunque el término aún no se usaba y no existía, esto fue la base de las redes neuronales artificiales [11].

Volviendo al *Asociacionismo*, es relevante mencionar que, además de David Hartley, Alexander Bain contribuyó con ideas fundamentales para la Regla de Aprendizaje de Hebbian. Esta regla establece una relación entre el proceso de memoria asociativa y la distribución de la actividad en grupos neuronales. Así describió una estructura que es muy parecida a la configuración actual de las redes neuronales: una neurona individual resume la estimulación de otra al seleccionar neuronas conectadas en el mismo grupo.

En 1949 se desarrolla la Regla de Aprendizaje de Hebbian, que viene de Donald O. Hebb. De aquí viene *Cells that fire together, wire together*, neuronas que se activan juntas, se conectan juntas.

"*Cuando un axón de la célula A está lo suficientemente cerca como para excitar una célula B y participa repetida o persistentemente en activarla, se produce algún proceso de crecimiento o cambio metabólico en una o ambas células de tal manera que, como eficiencia, como una de las células disparando B, se incrementa*"

Esto puede ser escrito matemáticamente y en un lenguaje de *machine learning* como:

$$\triangle \mathbf{w_i} = \eta \mathbf{x_i} \mathbf{y}$$

donde $\triangle \mathbf{w_i}$ es el campo de los pesos de la neurona $\mathbf{i}$, $\mathbf{y}$ es la respuesta postsináptica, la señal de entrada es $\mathbf{x_i}$, y η es la tasa de aprendizaje.

La Regla de Aprendizaje de Hebbian establece que la conexión entre dos unidades debe fortalecerse a medida que aumenta la frecuencia de las co-ocurrencias de estas dos unidades. El problema

es que a medida que aparecen más co-ocurrencias, los pesos de las conexiones siguen aumentando y los pesos de una señal dominante aumentarán exponencialmente. Esto se conoce como la inestabilidad de la Regla de Aprendizaje de Hebbian [17].

Posteriormente, en 1950, Turing plantea la pregunta *¿puede pensar una máquina?*. Presentando el *Test de Turing*, una prueba para determinar si una máquina es inteligente o no [18].

En 1956 fue utilizado el término inteligencia artificial por primera vez. Fue en la conferencia *Dartmouth Summer Research Project on Artificial Intelligence* de John McCarthy [19].

En 1959 Frank Rosenblatt desarrolla el perceptron. En vez de centrarse en los aspectos biológicos trató de construír un dispositivo electrónico. El pereptron es considerado la primera red neuronal artificial con capacidad de aprender. Se puede ver su arquitectura en la Figura 9 [12].

Más tarde, se desarrolla el modelo *Adaline*, siendo la primera red neuronal aplicada a un problema real, con la novedad de corregir los pesos durante el entrenamiento en base al error que cometía [20].

Al mismo tiempo, empiezan a aparecer algunos de los algoritmos más populares en *machine learning* y que permitieron trabajar con problemas de clasificación y reconocimiento de patrones.

En 1969 Minski y Papert estudian el poder de representación lineal. Esto es que el perceptron es fundamentalmente una función linear de señales de entrada. Por lo tanto, se limita a representar límites de decisión lineales como las operaciones lógicas NOT, AND u OR, sin embargo, no puede representar cuando se requiere un límite de decisión más sofisticado, como con la operación lógica XOR. Esto hizo que se perdiera el interés en las redes neuronales y las investigaciones en el campo de la inteligencia artificial y redes neuronales se vieran muy reducidas [21].

Sin embargo, algunos investigadores siguieron desarrollando el campo, es el caso de Paul Werbos, que en 1974 introduce lo que hoy se conoce como *backpropagation* cuando introduce la propagación hacia atrás de errores en el proceso de entrenamiento de una red neuronal artificial [22].

En los años 80 el interés por las redes neuronales artificiales vuelve y, en 1980, Kunihiko Fukushima establece una red neuronal que sirve de inspiración para las redes neuronales convulocionales, el *Neocognition* [23].

En 1986, David Rumelhart, Ronald Williams y Geoffrey Hinton publicaron un artículo en el que redescubrían el algoritmo *backpropagation*. En él demostraban que se podía entrenar una red neuronal con muchas capas ocultas de forma eficaz con este método y aprendez funciones no lineales. Así quedaban superadas las limitaciones del perceptron [24].

Yann LeCun en 1989 llevo a cabo una aplicación práctica del algoritmo *backpropagation* mostrando la posibilidad de llevar redes neuronales profundas a la práctica. A esta red la denominó *LeNet* que clasificaba dígitos escritos a mano [25].

En 2006, Geoffrey Hinton utiliza por primera vez el término *deep learning*. Así comienza la era del *deep learning*. Y en 2012 introdujo el *Dropout*, una manera eficiente de entrenar las redes neuronales.

En 2012, un grupo de estudiantes de doctorado con la supervisión de Geoffrey Hinton presentan la red neuronal concolucional *Alexnet* a la competición *Imagenet* [26]. Esta competición consiste en el etiquetado de imágenes con muchas clases e imágenes de entrenamiento. Esta red consiguió un *accuracy* del 63,3 % .

Actualmente, la mejor red sobre *ImageNet* es *CoCa*. Con un *accuracy* del 91 % [27].

A continuación, se repasará el estado actual del *deep learning*.

2.3. Estado actual

La arquitectura de red neuronal convolucional conocida como *AlexNet* marcó un hito significativo en el campo del aprendizaje profundo y la visión por computador. Fue presentada por un grupo de estudiantes de doctorado supervisados por Geoffrey Hinton en la competición *ImageNet* en 2012 [26]. AlexNet logró un impresionante rendimiento al obtener una precisión del 63.3 % en la clasificación de imágenes, superando a las propuestas anteriores. Este logro sentó las bases para una carrera en la investigación y el desarrollo de nuevas arquitecturas de Redes Neuronales Convolucionales (*Convolutional Neural Networks, CNN*) que buscan superar los resultados de AlexNet. Desde entonces, se han propuesto y desarrollado numerosas arquitecturas mejoradas, como VGGNet, GoogLeNet, ResNet y muchas otras, que han logrado una mayor precisión en el reconocimiento de imágenes y han ampliado la capacidad de las CNN para abordar una amplia variedad de problemas.

Además, la incorporación de nuevas técnicas de entrenamiento, como el aprendizaje por transferencia, que aprovecha los conocimientos previos de una red entrenada en una tarea relacionada, ha sido crucial. Esto permite que en lugar de entrenar desde cero, la red inicialmente aprenda características generales, como bordes, formas y texturas, que son útiles en muchas tareas de visión por computador. Luego, se realiza un ajuste fino en el conjunto de datos objetivo para adaptar estas características generales a las particularidades de la tarea específica, lo que acelera el proceso y supera la falta de datos. Esto ha contribuido significativamente a los avances en visión por computadora y reconocimiento de imágenes.

Además, las CNN han demostrado ser más precisas que los expertos humanos en tareas relacionadas con la visión como la detección de anomalías en imágenes satelitales o la presencia de daños en la infraestructura después de un desastre natural.

Hay muchos sistemas en los que se están usando algoritmos de *deep learning* hoy en día, algunos de los sistemas más recientes y que más impacto han tenido son:

- ***Copilot***: Asistente inteligente para ayudar a desarrollar código.
- ***Dalle-2***: Sistema para crear imágenes a partir de una descripción escrita.
- ***chatgpt***: Sistema de chat inteligente.

En los últimos años, el *deep learning* ha experimentado un gran avance en la investigación y en el desarrollo de nuevos algoritmos. Algunos de los más notables son:

- **Algoritmos de Convolutional Neural Networks (CNN)**: tareas de visión artificial y reconocimiento de imágenes.
- **Algoritmos de Recurrent Neural Networks (RNNs)**: procesamiento de secuencias de datos, como audio y texto.
- **Algoritmos de Generative Adversarial Networks (GANs)**: generación de imágenes y contenido de alta calidad a partir de datos de entrenamiento.
- **Algoritmos de Transformer**: han revolucionado el proceso de análisis de texto y han mejorado la eficiencia en tareas de traducción automática y resumen de texto.
- **Algoritmos de Reinforcement Learning**: permiten el aprendizaje automático de tareas complejas a través de la retroalimentación y la toma de decisiones basadas en recompensas.

El acceso abierto en Internet a contenido relacionado con el *deep learning* ha impulsado su rápido desarrollo al permitir una colaboración abierta y el acceso a diversos recursos y tecnologías, fomentando así la investigación y el avance en este campo.

En la Figura 8 se recoge el panorama del *Machine Learning* en la actualidad. Se puede apreciar cómo ésta tecnología está dando lugar a empresas que ayudan a su desarrollo. Algunas de las empresas más relevantes en el desarrollo del *Machine learning y Deep Learning* son NVIDIA, OpenAI o DeepMind.

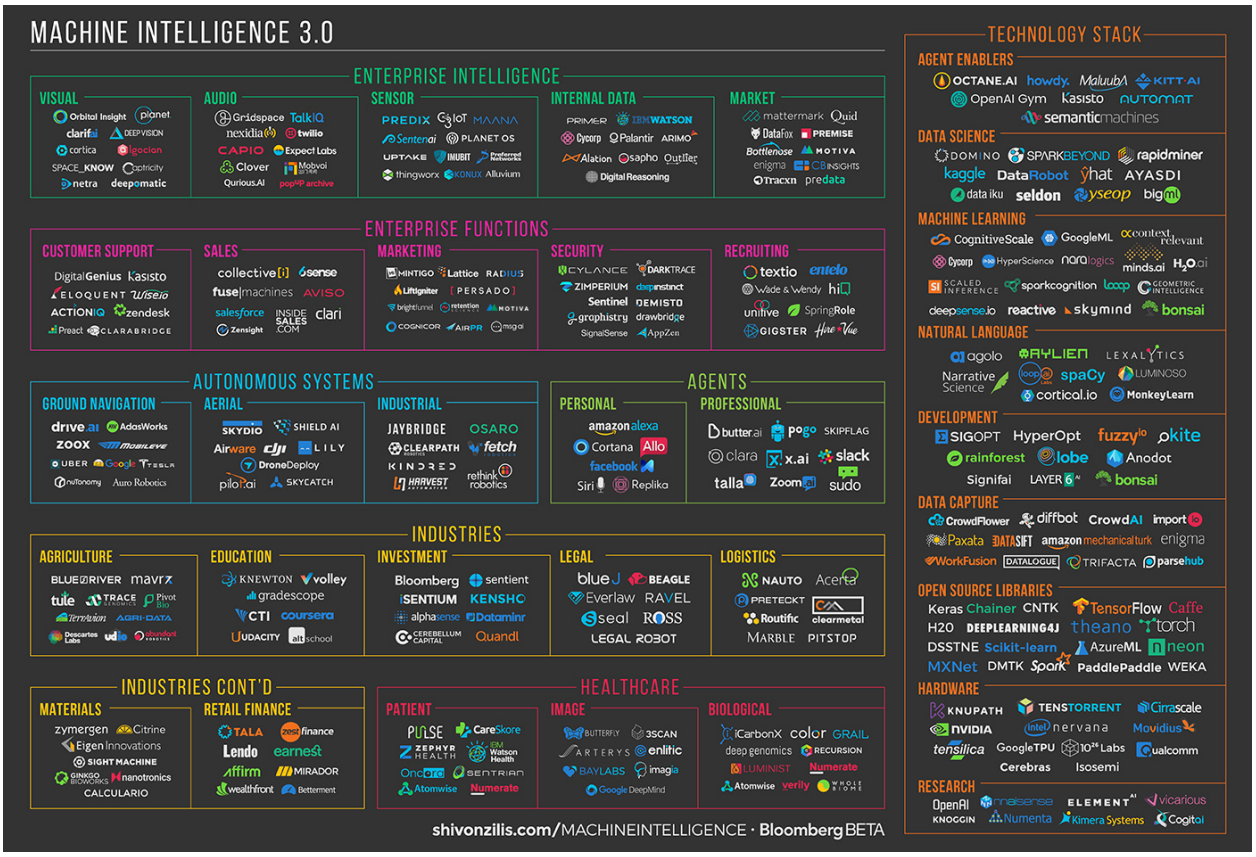

Figura 8: *Panorama del Machine Learning en la actualidad*

Fuente: [28]

Como se ha mencionado anteriormente, las redes neuronales requieren una gran capacidad de cómputo y de almacenamiento, es decir, *hardware*. Por eso el desarrollo de esta tecnología es también muy importante.

Durante los últimos años se ha vivido una explosión en el desarrollo de sistemas basados en *Deep Learning*. Aún así queda un largo recorrido para acelerar el entrenamiento de las redes y conseguir buenos resultados con menos datos de entrenamiento.

Aún no es posible conocer bien las limitaciones que presenta el *Deep Learning*, pero se está desarrollando de forma exponencial utilizando el funcionamiento del cerebro humano como inspiración para esta técnica, sin saber bien cómo funciona éste.

3. Redes Neuronales

En este capítulo se presentan el estudio de los conceptos teóricos necesarios para poder entender y utilizar las redes neuronales.

En su primer punto, una introducción a los fundamentos teóricos.

A continuación, en el segundo punto, se expondrá el desarrollo de redes neuronales, con sus etapas.

3.1. Fundamentos Teóricos

Las redes neuronales artificiales son un modelo computacional inspirado en el funcionamiento del cerebro humano y su sistema nervioso. Estas redes están compuestas por unidades de procesamiento interconectadas, llamadas neuronas artificiales o nodos, que trabajan en conjunto para resolver problemas complejos de aprendizaje automático.

El objetivo fundamental de una red neuronal es aprender a partir de los datos para realizar tareas como clasificación, reconocimiento de patrones, predicción, entre otras. A través del proceso de entrenamiento, la red neuronal ajusta los pesos y las conexiones entre las neuronas, de manera que pueda realizar inferencias y tomar decisiones basadas en la información que recibe.

Una de las características clave de las redes neuronales es su capacidad para aprender de forma no lineal y adaptativa. A diferencia de los enfoques tradicionales de programación, donde se definen reglas y algoritmos específicos, las redes neuronales pueden aprender automáticamente a partir de datos y generar modelos que generalizan a datos no vistos previamente.

En esta sección, exploraremos los fundamentos de las redes neuronales, comenzando con el Perceptrón como uno de los modelos más simples. Analizaremos su estructura, su proceso de entrenamiento y sus aplicaciones iniciales. Posteriormente, nos adentraremos en arquitecturas más complejas, como las redes neuronales convolucionales o las redes recurrentes, que han demostrado un gran rendimiento en tareas de visión por computador. También discutiremos la importancia de las funciones de activación y otras técnicas utilizadas en el diseño y entrenamiento de redes neuronales.

3.1.1. Perceptrón

El Perceptrón es uno de los modelos más básicos y fundamentales en el campo de las redes neuronales. Fue inicialmente propuesto por McCulloch y Pitts en el año 1943 [11], como un modelo matemático de una neurona artificial, inspirandose en la estructura y funcionamiento de la neurona biológica. En honor a sus creadores, también se le conoce como la neurona de McCulloch-Pitts. Posteriormente, Frank Rosenblatt [12], realiza aproximación matemática pero tratando de construír un dispositivo electrónico. Se le considera el primer modelo de neurona artificial. Aunque es un modelo simple, sentó las bases para el desarrollo posterior de redes neuronales más complejas.

El perceptrón es un modelo de clasificación binaria que toma varias entradas y produce una única salida. Cada entrada se multiplica por un peso correspondiente, y luego se realiza una suma ponderada de estas entradas ponderadas. Los pesos de las neuronas representan la importancia o influencia que cada entrada tiene en la salida del perceptrón. La salida del perceptrón depende de si el resultado de la suma ponderada supera o no un umbral establecido. Esta salida binaria se

utiliza para clasificar los datos de entrada en dos categorías distintas [12].

A continuación se presenta un ejemplo básico para ilustrar el funcionamiento del perceptrón basado en el perceptrón de la Figura 9.

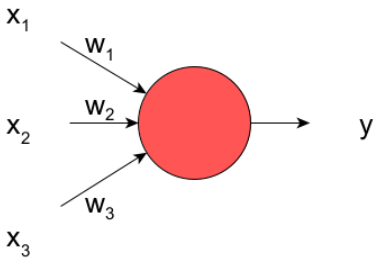

Figura 9: *Perceptrón*

Fuente: Elaboración Propia

Supongamos que tenemos un perceptrón con las siguientes entradas y pesos:

$$x1 = 1 \qquad w1 = 0{,}5 \qquad x2 = 0 \qquad w2 = -1 \qquad x3 = 1 \qquad w3 = 0{,}8$$

Además, consideremos que el umbral θ es igual a 0.2.

1. Se proporcionan las entradas binarias: $x1 = 1$, $x2 = 0$, $x3 = 1$.
2. Se multiplican las entradas por sus respectivos pesos: $(1 \times 0{,}5) + (0 \times -1) + (1 \times 0{,}8) = 1{,}3$.
3. La suma ponderada total es 1.3.
4. Se compara la suma ponderada con el umbral: $1{,}3 \geq 0{,}2$. Como se cumple esta condición, la neurona se activa y produce una salida de 1.
5. Por lo tanto, el resultado de y es una clasificación binaria, donde $y = 1$.

A pesar de su simplicidad, el Perceptrón ha demostrado ser efectivo en una variedad de problemas de clasificación linealmente separables. Sin embargo, tiene limitaciones significativas, ya que no puede manejar problemas no lineales ni aprender funciones complejas. Estas limitaciones dieron paso al desarrollo de arquitecturas más complejas, como las redes convolucionales, o las recurrentes, que permiten el aprendizaje de representaciones más sofisticadas y la resolución de problemas más desafiantes.

En las siguientes secciones, se explorarán las funciones de activación, que son fundamentales para el funcionamiento de las redes neuronales, así como las arquitecturas más complejas que han surgido a partir del Perceptrón.

3.1.2. Funciones de Activación

En el campo de las redes neuronales, tanto el perceptrón como las funciones de activación desempeñan un papel fundamental en el funcionamiento de estos modelos. Mientras que el perceptrón

es un modelo básico de clasificación binaria que utiliza una combinación lineal de entradas y pesos para producir una salida, las funciones de activación permiten introducir no linealidad en el modelo, lo cual resulta crucial para el aprendizaje y la representación de relaciones complejas entre variables de entrada y salida [12].

Aunque el perceptrón es efectivo en problemas de clasificación linealmente separables, presenta limitaciones para manejar problemas no lineales y aprender funciones complejas. La introducción de funciones de activación en las redes neuronales es lo que permite superar estas limitaciones al agregar capacidades de modelado no lineal.

Una red neuronal es un modelo más complejo que consiste en la interconexión de múltiples perceptrones y capas de neuronas. Cada neurona en una red neuronal recibe entradas ponderadas y aplica una función de activación a la suma ponderada. Las funciones de activación se encargan de introducir no linealidad en cada neurona, lo cual permite que la red pueda aprender representaciones más sofisticadas y resolver problemas no lineales [10].

Por eso las funciones de activación son esenciales en las redes neuronales, incluyendo el perceptrón, ya que permiten la incorporación de no linealidad en el modelo, lo cual es crucial para aprender y representar relaciones complejas entre variables de entrada y salida. Estas funciones juegan un papel fundamental en el funcionamiento de las redes neuronales, permitiendo que sean capaces de resolver una amplia gama de problemas, incluyendo aquellos que no son linealmente separables.

Existen diferentes tipos de funciones de activación utilizadas en las redes neuronales, cada una con sus propias características y propiedades. A continuación, se presentan algunas de las funciones de activación más comunes:

Función identidad La función identidad, también conocida como función lineal, es una de las funciones de activación más simples. Esta función mapea directamente la entrada a la salida sin realizar ninguna transformación. Matemáticamente, se define como:

$$f(x) = x \tag{1}$$

La función identidad es comúnmente utilizada en capas de salida cuando se desea obtener una salida sin ninguna modificación adicional. Además, esta función es diferenciable en todo su dominio, lo que la hace compatible con algoritmos de aprendizaje basados en el cálculo de gradientes.

Función Escalón La función escalón, también conocida como función de Heaviside, es una función de activación binaria que asigna un valor de salida de 1 si la entrada es mayor o igual a un umbral definido, y un valor de salida de 0 si la entrada es menor que ese umbral. Matemáticamente, se define como:

$$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \tag{2}$$

La función escalón es una de las funciones de activación más simples y fue utilizada en los primeros modelos de perceptrón. Sin embargo, esta función no es diferenciable en el punto de transición, lo que dificulta el uso de algoritmos de optimización basados en gradientes.

Función Sigmoide La función sigmoide, es una función suave que produce una salida continua en el rango de 0 a 1. Matemáticamente, se define como:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

La función sigmoide es utilizada principalmente en problemas de clasificación binaria, donde se busca asignar una probabilidad a cada clase. Sin embargo, esta función tiene algunas limitaciones, como la tendencia a saturar en los extremos de su rango, lo que puede dificultar el entrenamiento de la red.

Función Tangente Hiperbólica (*Tanh*) La función tangente hiperbólica, también conocida como tanh, es una función de activación que mapea la entrada a un rango entre -1 y 1. Matemáticamente, se define como:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4}$$

La función tanh es una versión escalada y desplazada de la función sigmoide, y al igual que esta última, es útil en problemas de clasificación binaria. A diferencia de la función sigmoide, la función tanh tiene un rango simétrico y puede tomar valores negativos, lo que la hace adecuada para problemas en los que se requiere una respuesta tanto positiva como negativa.

Función Unidad Lineal Rectificada (*Rectified Linear Unit* (*ReLU*)) La función ReLU es una función no lineal ampliamente utilizada en redes neuronales. Esta función produce una salida de 0 si la suma ponderada de las entradas y los pesos es menor que cero, y una salida lineal igual a la entrada si es mayor o igual a cero. Matemáticamente, se define como:

$$f(x) = \text{máx}(0, x) \tag{5}$$

La función ReLU es especialmente útil en redes neuronales profundas debido a su capacidad para superar el problema del desvanecimiento del gradiente. Además, esta función es fácil de calcular y no tiene restricciones en su rango de valores.

Función Softmax La función *softmax* es una función de activación utilizada comúnmente en la capa de salida de una red neuronal para problemas de clasificación multiclase. Esta función toma como entrada un vector de valores reales y produce un vector de probabilidades, donde cada elemento del vector representa la probabilidad de pertenecer a una clase específica. Matemáticamente, la función *softmax* se define como:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{6}$$

La función softmax garantiza que la suma de todas las probabilidades sea igual a 1, lo que permite interpretar las salidas de la red como distribuciones de probabilidad. Esto facilita la selección de la clase con mayor probabilidad como la predicción final. Se utiliza para transformar las salidas lineales de la red en probabilidades para cada clase [10].

3.1.3. Arquitecturas

En el campo de las redes neuronales, existen varios tipos de arquitecturas que se han desarrollado para abordar diferentes tipos de problemas y aprovechar las características específicas de los datos. A continuación, se presentan algunos de los tipos de arquitecturas ampliamente utilizadas.

Estas arquitecturas se pueden diferenciar, según el enfoque para el problema a solucionar, por:

- **Capas de la red**
- **Número de capas**
- **Número de neuronas que hay en cada capa**
- **Tipo de neuronas que hay en cada capa**

A continuación, en la Figura 10 se presentan algunos tipos comunes de redes neuronales junto con una breve explicación de cada uno:

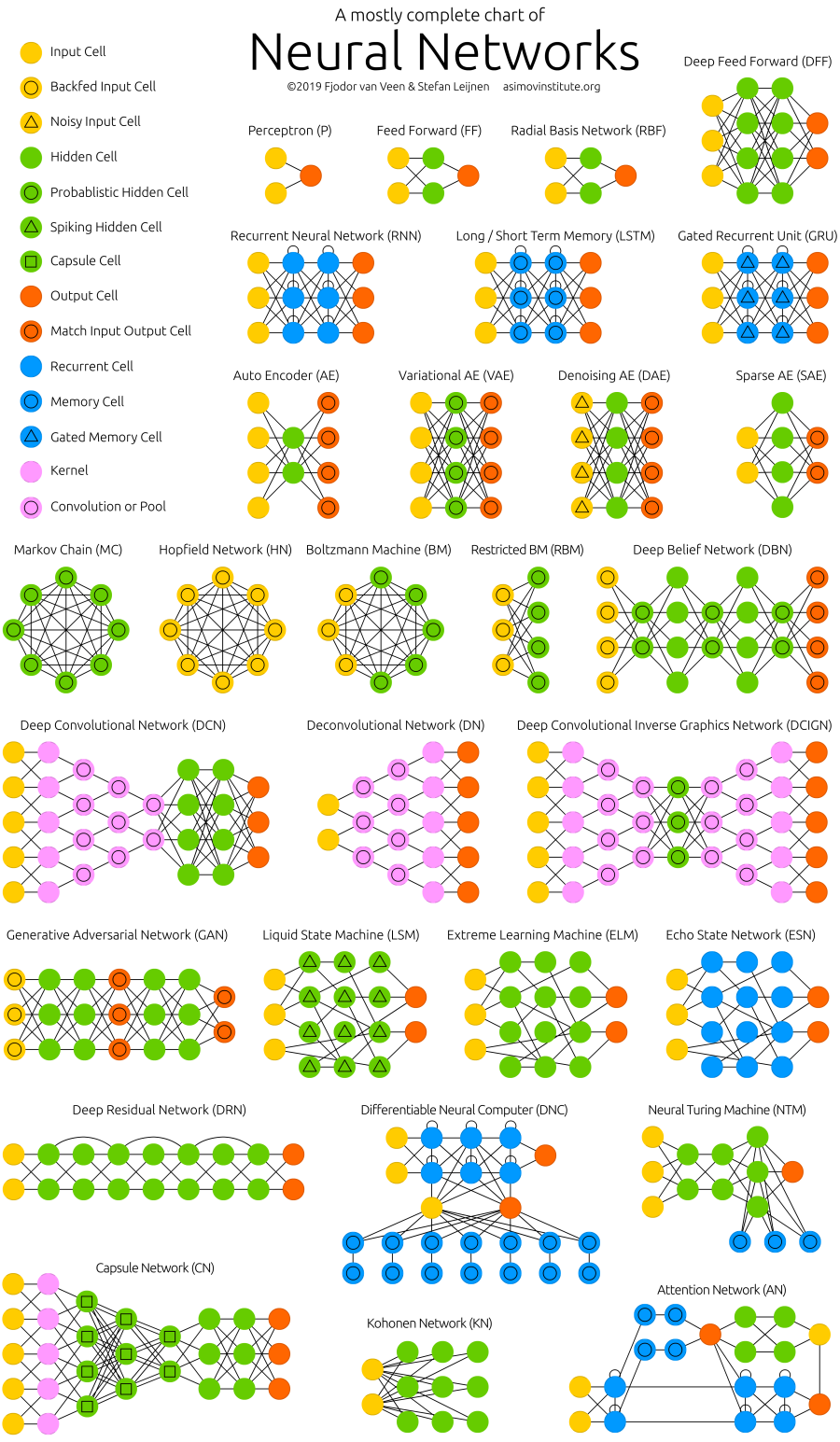

Figura 10: Arquitecturas de Redes Neuronales

Fuente: The mostly complete chart of Neural Networks, explained by Andrew Tch

- **Redes Neuronales Recurrentes (*Recurrent Neural Network (RNN)*)**: Las Redes Neuronales Recurrentes (RNN) son arquitecturas diseñadas para modelar datos secuenciales, como texto o series temporales. Las RNN tienen conexiones recurrentes que les permiten mantener una memoria interna, lo que las hace adecuadas para tareas de procesamiento secuencial. Las RNN tienen una distribución de capas en forma de bucle, donde la salida de una capa se retroalimenta como entrada en la siguiente capa. Las capas en una RNN pueden ser recurrentes, utilizando una función de activación como la función de activación recurrente (tanh) o la unidad LSTM (Long Short-Term Memory). Las RNN se utilizan en aplicaciones como el procesamiento de lenguaje natural, la traducción automática y la generación de texto [10]. En la Figura 11 se muestra un ejemplo de una RNN.

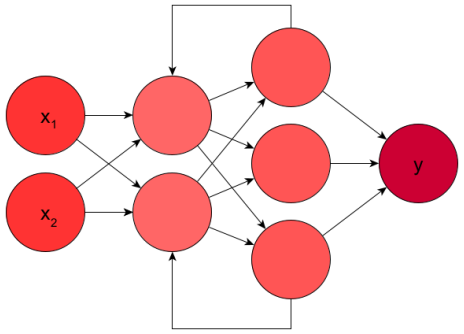

Figura 11: *Recurrent Neural Network*

Fuente: Elaboración Propia

- ***Auto Encoder (AE)***: Los Autoencoders (AE) son arquitecturas utilizadas para el aprendizaje no supervisado y la reducción de dimensionalidad. Están compuestos por dos partes principales: el codificador, que transforma los datos de entrada en una representación latente de menor dimensión, y el decodificador, que reconstruye los datos de entrada a partir de la representación latente [29]. Los AE tienen una distribución de capas en forma de "V´´, donde las capas de la parte codificadora disminuyen gradualmente la dimensionalidad, mientras que las capas de la parte decodificadora aumentan la dimensionalidad. Las capas en un AE pueden utilizar diversas funciones de activación, como la función ReLU en las capas ocultas y la función sigmoide en la capa de salida. Los AE se utilizan en aplicaciones como la detección de anomalías, la compresión de datos y la generación de imágenes [30]. En la Figura 12 se muestra una posible estructura de un AE.

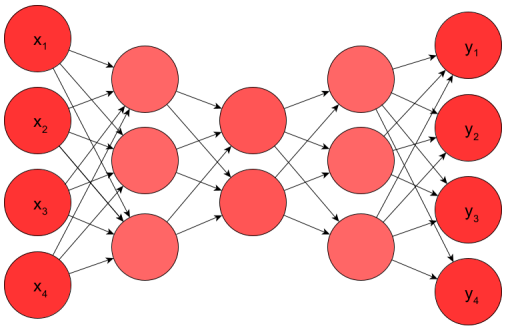

Figura 12: *Autoencoder*

Fuente: Elaboración Propia

- **Red Adversaria Generativa (*Generative Adversarial Network (GAN)*)**: Las GAN son arquitecturas utilizadas para generar nuevos datos sintéticos que se asemejen a un conjunto de datos de entrenamiento dado. Constan de dos partes: el generador, que crea nuevos datos, y el discriminador, que intenta distinguir entre los datos reales y los datos generados. Ambas partes se entrenan de forma adversarial, mejorando continuamente sus habilidades a medida que compiten entre sí. Pueden tener una distribución de capas similar a los AE, con capas de codificación y decodificación en el generador, y capas de discriminación en el discriminador. Las capas pueden utilizar funciones de activación como la función ReLU en las capas ocultas y la función sigmoide en la capa de salida del discriminador. Las GAN se utilizan en aplicaciones como la generación de imágenes realistas, la mejora de imágenes y la síntesis de voz [31]. En la Figura 13 se muestra una posible estructura de una GAN.

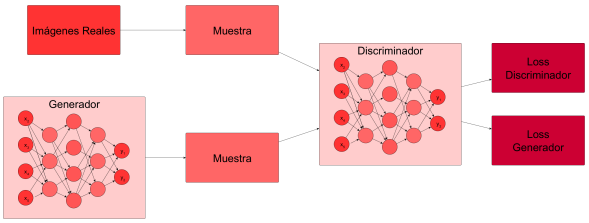

Figura 13: *Generative Adversarial Network*

Fuente: Elaboración Propia

- **Redes Neuronales Convolucionales (*Convolutional Neural Network (CNN)*)**: Las CNN son arquitecturas diseñadas específicamente para el procesamiento de datos estructurados, como imágenes y videos. Se basan en la idea de utilizar convoluciones en lugar de operaciones matriciales tradicionales para extraer características espaciales de los datos. Estas redes se caracterizan por tener capas convolucionales, capas de agrupación (*pooling*) y capas completamente conectadas. Las capas convolucionales aplican filtros a las regiones locales de los datos de entrada, capturando patrones visuales como bordes y texturas. Las capas de agrupación reducen la dimensionalidad espacial de los mapas de características. Por último, las capas completamente conectadas realizan la clasificación o regresión final basándose en las características extraídas. Las funciones de activación comúnmente utilizadas en las CNN incluyen ReLU en las capas convolucionales y sigmoide o softmax en la capa de salida, dependiendo del tipo de problema. Las CNN son muy utilizadas en aplicaciones de reconocimiento de imágenes, detección de objetos, segmentación semántica y análisis de video [10]. En la Figura 14 se muestra una posible representación de una CNN.

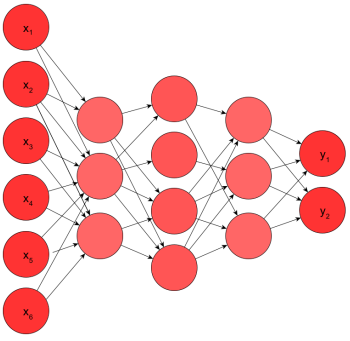

Figura 14: *Convolutional Neural Network*

Fuente: Elaboración Propia

Estas son solo algunas de las arquitecturas de redes neuronales más comunes utilizadas en diversos campos. Cada una de ellas ofrece un enfoque único para abordar diferentes tipos de problemas y aprovechar las características específicas de los datos. La elección de la arquitectura adecuada depende del tipo de problema, los datos disponibles y los objetivos de la aplicación. Dado que este Trabajo de Fin de Grado consiste en un problema enfocado en el campo de la visión por computador, se hará uso de las *Convolutional Neural Networks*.

3.1.4. Casos de clasificación

En el campo del aprendizaje automático, existen diferentes enfoques para realizar la clasificación de datos. A continuación, se presentan una técnica común utilizada para la clasificación: la codificación *one-hot*.

Codificación One-Hot La codificación *one-hot* es una técnica utilizada para representar las etiquetas de clase en un problema de clasificación. Consiste en transformar las etiquetas de clase en vectores binarios de dimensión igual al número de clases. En este esquema de codificación, solo un elemento del vector tiene el valor 1, correspondiente a la clase correcta, mientras que todos los demás elementos tienen el valor 0. Esta representación permite que la red neuronal trabaje con las etiquetas de manera más efectiva, ya que cada clase se trata de forma independiente y se evita cualquier suposición de orden o relación entre las clases. Según Bengio (2009), el one-hot encoding se utiliza para representar las etiquetas de clase como vectores binarios para el entrenamiento de redes neuronales [30].

Esta técnica, la codificación *one-hot*, es fundamental en el campo de la clasificación en aprendizaje automático y se utiliza en diversas aplicaciones para asignar probabilidades a clases.

3.1.5. Aplicación a imágenes

La aplicación de redes neuronales en el campo de la visión por computador ha tenido un gran impacto en problemas relacionados con el procesamiento y el análisis de imágenes. Una de las etapas más importantes en el procesamiento de imágenes es la transformación de la estructura espacial de la imagen en un formato adecuado para su entrada a la red. Una técnica comúnmente utilizada en este proceso es el *flattening*.

Flattening El *flattening* es el proceso de convertir una imagen bidimensional en un vector unidimensional. Si se considera una imagen en color de dimensiones $M \times N \times 3$, donde M representa el número de píxeles en la altura, N representa el número de píxeles en el ancho, y 3 corresponde a los tres canales de color (rojo, verde y azul). Para aplicar el *flattening*, la imagen se reorganiza en un vector de longitud $M \times N \times 3$, donde los píxeles se concatenan en un solo vector siguiendo un orden específico.

Este proceso es necesario para alimentar la imagen como entrada a una red neuronal de tipo densamente conectada, donde cada píxel de la imagen se convierte en una característica independiente. De esta manera, la estructura espacial de la imagen se pierde transformándose en una representación lineal de los píxeles.

Es importante destacar que el *flattening* no implica la pérdida total de la información espacial. Las capas posteriores de la red neuronal pueden aprender a extraer características espaciales y patrones relevantes a partir del vector unidimensional. Es más, esta pérdida de información espacial

se puede mitigar mediante el uso de arquitecturas de redes neuronales específicas, como las CNN, que son especialmente adecuadas para procesar datos de imagen y preservar la estructura espacial [10].

El *flattening* es una técnica fundamental en el procesamiento de imágenes en redes neuronales y permite el uso eficiente de algoritmos de aprendizaje automático en problemas de visión por computador. Aunque implica una pérdida de la información espacial, es una estrategia práctica y efectiva para representar imágenes como vectores de características y aprovechar el poder de las redes neuronales en el análisis de imágenes.

3.2. Desarrollo de Redes Neuronales

El desarrollo de redes neuronales implica varias etapas clave que permiten construir y entrenar modelos efectivos tal y como se adelantó en la sección **??**. A continuación, se describirán las etapas principales del desarrollo de redes neuronales, así como los aspectos generales de cada una de ellas y su objetivo.

3.2.1. Etapas de desarrollo

En el desarrollo de redes neuronales, se suelen seguir tres etapas principales: entrenamiento, validación y test. Estas etapas son fundamentales para evaluar el desempeño y la generalización del modelo.

- **Entrenamiento**: En esta etapa, el modelo se expone a un conjunto de datos de entrenamiento y se ajustan los pesos y los sesgos de las neuronas a través del algoritmo de optimización seleccionado. El objetivo del entrenamiento es lograr que el modelo aprenda a mapear las entradas a las salidas deseadas y minimizar la función de pérdida [10]. Durante el entrenamiento, se utilizan diferentes técnicas, como la retropropagación del error (backpropagation), para ajustar los parámetros del modelo.
- **Validación**: Después del entrenamiento, se realiza la validación utilizando un conjunto de datos separado llamado conjunto de validación. El propósito de la validación es evaluar la capacidad de generalización del modelo y ajustar los hiperparámetros para evitar el sobreajuste (overfitting) [10]. En esta etapa, se monitorea el desempeño del modelo en el conjunto de validación y se realizan ajustes adicionales si es necesario.
- **Test**: Una vez que se ha realizado el entrenamiento y la validación, se evalúa el rendimiento final del modelo en un conjunto de datos completamente independiente llamado conjunto de prueba. Este conjunto de datos no se utiliza en ninguna etapa del entrenamiento o validación, y proporciona una evaluación objetiva del rendimiento del modelo en datos no vistos previamente [10]. El rendimiento en el conjunto de prueba es un indicador de la capacidad del modelo para generalizar y aplicar el aprendizaje a nuevos datos.

Es importante tener en cuenta que los datos utilizados en cada etapa deben prepararse adecuadamente. Durante el entrenamiento, se utiliza el conjunto de datos de entrenamiento para ajustar los parámetros del modelo. En la etapa de validación, se emplea un conjunto de datos separado para evaluar el rendimiento y ajustar los hiperparámetros. Finalmente, en la etapa de prueba, se utiliza un conjunto de datos completamente independiente para la evaluación final del modelo. La preparación de datos incluye tareas como la división de los datos en conjuntos de entrenamiento, validación y prueba, la normalización de los datos y el manejo adecuado de los valores atípicos [32].

3.2.2. Etapa de entrenamiento

En la etapa de entrenamiento de una red neuronal, se llevan a cabo varios aspectos importantes que influyen en el rendimiento y la convergencia del modelo. El rendimiento del modelo se refiere a su capacidad para realizar correctamente la tarea para la cual ha sido diseñado, produciendo resultados precisos y consistentes en nuevos datos. La convergencia del modelo se refiere a la estabilización de los pesos y los resultados a medida que avanza el entrenamiento, alcanzando un estado óptimo donde la función de pérdida se ha minimizado y se han capturado los patrones importantes de los datos. Para lograr un buen rendimiento y convergencia, es necesario considerar aspectos como la arquitectura de la red, la inicialización de los pesos, la selección de la función de pérdida, los

algoritmos de optimización y los hiperparámetros del modelo. Estos aspectos interactúan entre sí y deben ser ajustados de manera adecuada para obtener los mejores resultados en la tarea específica. El proceso de entrenamiento de una red neuronal es iterativo y requiere experimentación y ajuste fino para lograr un rendimiento óptimo y una convergencia satisfactoria [10].

Función de pérdida (Estimación del error). Durante el entrenamiento, es necesario definir una función de pérdida que mida la discrepancia entre las salidas predichas por el modelo y las salidas deseadas. Esta función de pérdida cuantifica el error del modelo y es fundamental para el cálculo del gradiente y la actualización de los pesos durante la retropropagación [10]. La cuantificación del error del modelo implica medir la diferencia entre las salidas predichas y las salidas deseadas, proporcionando una medida numérica que indica qué tan bien está realizando el modelo en términos de precisión.

El cálculo del gradiente sucede en la retropropagación (*backpropagation*), ya que representa la tasa de cambio de la función de pérdida con respecto a los pesos del modelo. El gradiente se utiliza para determinar la dirección en la que deben ajustarse los pesos para reducir el error del modelo. A través de la actualización de los pesos en dirección opuesta al gradiente, se busca minimizar la función de pérdida y mejorar el rendimiento del modelo.

La tasa de aprendizaje (*learning rate*) es un parámetro crítico en el entrenamiento de la red neuronal. Representa la velocidad a la que los pesos se ajustan durante su actualización. Una tasa de aprendizaje alta puede permitir un rápido ajuste de los pesos, pero también puede llevar a oscilaciones y dificultar la convergencia, mientras que una tasa de aprendizaje baja puede generar una convergencia más lenta. La elección adecuada de la tasa de aprendizaje es esencial para lograr un entrenamiento efectivo y asegurar una convergencia estable hacia un mínimo óptimo de la función de pérdida [10].

En resumen, durante el entrenamiento de una red neuronal, la función de pérdida cuantifica el error del modelo, el cálculo del gradiente determina la dirección de ajuste de los pesos, y la tasa de aprendizaje controla la velocidad de actualización de los pesos en función del gradiente calculado. Estos aspectos son fundamentales para el ajuste de la red y su capacidad para aprender y adaptarse a los datos de entrenamiento.

Algunos ejemplos de funciones de pérdida comunes incluyen:

- **Error cuadrático medio (MSE)**: Calcula la diferencia cuadrática media entre las salidas predichas y las salidas deseadas. Es una métrica utilizada en los problemas de regresión.
- **Entropía cruzada (Cross-entropy)**: Mide la divergencia entre la distribución de probabilidad predicha y la distribución de probabilidad real. Es ampliamente utilizada en problemas de clasificación.
- **Logaritmo de verosimilitud negativa (Negative log-likelihood)**: Es similar a la entropía cruzada y se utiliza especialmente en problemas de clasificación con salidas probabilísticas.
- **Pérdida Hinge (Hinge loss)**: Es utilizada en problemas de clasificación con margen, como en las máquinas de vectores de soporte (SVM).

- **Pérdida de entropía cruzada categórica (Categorical cross-entropy loss)**: Es una variante de la entropía cruzada utilizada en problemas de clasificación multiclase con salidas categóricas.

Inicialización de pesos. La inicialización de los pesos de las neuronas en una red neuronal es otro de los pasos críticos para evitar problemas de convergencia y asegurar un entrenamiento eficiente. Diferentes técnicas de inicialización han sido propuestas, como la inicialización de Xavier y la inicialización de He, que buscan establecer valores iniciales adecuados para los pesos. Además, el *Transfer Learning* es una técnica que aprovecha los conocimientos previamente aprendidos. Esta técnica puede ayudar a mejorar el rendimiento y acelerar el entrenamiento de una red neuronal.

- **Inicialización de Xavier**: Esta técnica busca establecer valores iniciales adecuados para los pesos de las neuronas. Se basa en considerar tanto la cantidad de neuronas de entrada como la cantidad de neuronas de salida de cada capa para ajustar la escala de los pesos iniciales [33].
- **Inicialización de He**: Esta técnica es una variante de la inicialización de Xavier. Se utiliza principalmente en redes neuronales con funciones de activación rectificadas linealmente (ReLU) y ajusta la escala de los pesos iniciales teniendo en cuenta únicamente la cantidad de neuronas de entrada.
- **Transfer learning**: El transfer learning es una técnica que aprovecha los conocimientos previamente aprendidos por una red neuronal entrenada en un dominio relacionado y los aplica a una nueva tarea o dominio [34]. En lugar de entrenar una red neuronal desde cero, se utilizan los pesos pre-entrenados como punto de partida, lo que puede acelerar el entrenamiento y mejorar el rendimiento en tareas similares o relacionadas.

Algoritmos de optimización. Durante el entrenamiento, se utiliza un algoritmo de optimización para ajustar los pesos y reducir la función de pérdida. El descenso por gradiente es uno de los algoritmos más comunes, que utiliza el cálculo del gradiente para actualizar los pesos en la dirección opuesta al gradiente [10]. Además, existen variantes del descenso por gradiente que incorporan técnicas como el momento, Adagrad, RMSprop y Adam, que buscan acelerar la convergencia y mejorar la capacidad de adaptación del modelo a diferentes tipos de datos y tareas [35].

A continuación, se presentan breves descripciones de cada uno de estos algoritmos de optimización:

- **Descenso por gradiente**: Este algoritmo actualiza los pesos en función del gradiente de la función de pérdida. El tamaño del paso de actualización está determinado por la tasa de aprendizaje.
- **Descenso por gradiente con momento**: Esta variante del descenso por gradiente incorpora un término de momento que acumula los gradientes anteriores para determinar la dirección y el tamaño del paso de actualización. Esto ayuda a acelerar el proceso de convergencia y a evitar oscilaciones en el espacio de búsqueda.
- **Adagrad**: Este algoritmo adapta la tasa de aprendizaje para cada parámetro en función de la frecuencia con la que se ha actualizado anteriormente. Esto permite un ajuste más agresivo para parámetros poco frecuentes y un ajuste más conservador para parámetros frecuentes.

- **RMSprop**: Similar a Adagrad, RMSprop también adapta la tasa de aprendizaje para cada parámetro. Sin embargo, utiliza una media móvil de los gradientes cuadrados para normalizar la tasa de aprendizaje y evitar que se vuelva demasiado pequeña.
- **Adam**: Adam combina las ideas del descenso por gradiente con momento y RMSprop. Utiliza momentos de primer y segundo orden para adaptar la tasa de aprendizaje y ajustar los pasos de actualización en función de la magnitud de los gradientes y la media móvil de los gradientes cuadrados.

Estos algoritmos de optimización desempeñan un papel fundamental en el desarrollo de redes neuronales, ya que influyen en la velocidad de convergencia y en la calidad de los resultados obtenidos durante el entrenamiento. La elección del algoritmo adecuado puede depender del tipo de problema, del tamaño del conjunto de datos y de otros factores específicos de la tarea a realizar por el modelo.

Tipos de optimización. Durante el entrenamiento de una red neuronal, es común utilizar lotes de entrenamiento (*batches*) que consiste en dividir el conjunto completo de datos de entrenamiento en lotes de tamaño fijo y realiza el cálculo del gradiente y la actualización de los pesos para cada lote. [10].

A continuación de explican diferentes tipos de optimización:

- **Optimización por lotes (batch training)**: Este enfoque utiliza el conjunto completo de datos de entrenamiento en cada paso de actualización. El gradiente se calcula promediando los gradientes individuales de todos los datos en el lote y luego se actualizan los pesos en función de este gradiente promedio. Aunque computacionalmente costoso, el entrenamiento por lotes puede ofrecer una convergencia más estable y precisa [36]. El valor del lote en este caso sería el tamaño del conjunto de datos.
- **Optimización estocástica (stochastic optimization)**: En este enfoque, los pesos se actualizan después de cada dato de entrenamiento. En lugar de calcular el gradiente promedio para todo el conjunto de datos, se calcula el gradiente individualmente para cada dato y se actualizan los pesos en función de ese gradiente. Esto puede ser más eficiente en términos de tiempo de entrenamiento, pero puede introducir una mayor variabilidad en la convergencia debido a la naturaleza aleatoria de los datos seleccionados. El tamaño del lote es 1, pues se actualiza después de cada dato de entrenamiento individual.
- **Optimización en minilotes (mini-batch optimization)**: Este enfoque es un compromiso entre el entrenamiento por lotes y la optimización estocástica. En lugar de utilizar el conjunto completo de datos o un solo dato, se selecciona un subconjunto de datos de entrenamiento de tamaño fijo, llamado minilote. El gradiente se calcula promediando los gradientes individuales de los datos en el minilote y se actualizan los pesos en función de este gradiente promedio. El tamaño del minilote es un hiperparámetro que se puede ajustar y suele ser un valor intermedio entre el tamaño del conjunto de datos completo y 1.

La elección del tipo de optimización depende del tamaño del conjunto de datos, de los recursos computacionales disponibles y de la naturaleza del problema. Cada enfoque tiene sus ventajas y desventajas en términos de tiempo de entrenamiento, estabilidad y capacidad para generalizar a nuevos datos.

3.2.3. Etapa de entrenamiento: Hiperparámetros

La etapa de entrenamiento de una red neuronal involucra la configuración de varios hiperparámetros que afectan el proceso de aprendizaje y la capacidad de generalización del modelo. Algunos de los hiperparámetros más importantes incluyen:

- **Arquitectura**: Se refiere a la estructura y diseño de la red neuronal, incluyendo el número de capas ocultas, el número de neuronas en cada capa y la conexión entre ellas. La elección de una arquitectura adecuada depende del tipo de problema y los datos disponibles.
- **Inicialización de pesos**: Los pesos de las conexiones entre las neuronas se inicializan con valores aleatorios antes del entrenamiento. Diferentes técnicas de inicialización, como la inicialización de Xavier, He y transfer learning, se utilizan para establecer valores iniciales adecuados y facilitar la convergencia del modelo [33].
- **Número de épocas**: Representa el número de veces que el conjunto completo de datos de entrenamiento se presenta a la red neuronal durante el entrenamiento. Un número insuficiente de épocas puede resultar en un modelo subentrenado, mientras que un número excesivo puede llevar al sobreajuste.
- **Tamaño de lotes**: Indica la cantidad de datos de entrenamiento que se utilizan para calcular el gradiente y actualizar los pesos en cada paso de entrenamiento. El tamaño del lote afecta la eficiencia computacional y la estabilidad del entrenamiento [35].
- **Elección de algoritmo de optimización**: Seleccionar el algoritmo de optimización adecuado es crucial para mejorar el rendimiento del modelo. Algunos algoritmos populares incluyen el descenso por gradiente, descenso por gradiente con momento, Adagrad, RMSprop y Adam. Cada algoritmo tiene sus propias características y ajustes de parámetros [35].
- **Parámetros del algoritmo de optimización**: Cada algoritmo de optimización tiene parámetros específicos que deben configurarse, como la tasa de aprendizaje, el momento, el coeficiente de decaimiento, entre otros. Estos parámetros controlan la velocidad de convergencia y la adaptabilidad del modelo durante el entrenamiento.
- **Función de pérdida**: La función de pérdida cuantifica la discrepancia entre las salidas predichas por el modelo y las salidas deseadas. Algunas funciones de pérdida comunes incluyen el error cuadrático medio (MSE), la entropía cruzada y el logaritmo de verosimilitud negativa. La elección de la función de pérdida depende del tipo de problema y las salidas deseadas [10].

La configuración adecuada de estos hiperparámetros es esencial para lograr un buen rendimiento y evitar problemas como el sobreajuste o el subajuste durante el entrenamiento de la red neuronal.

3.2.4. Etapa de entrenamiento: Control y Seguimiento

Durante la etapa de entrenamiento de una red neuronal, es importante tener mecanismos de control y seguimiento que permitan mejorar el rendimiento y evitar problemas como el sobreajuste o el subajuste.

El sobreajuste, también conocido como sobreentrenamiento, ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento y pierde su capacidad de generalización. Esto significa que el modelo puede memorizar los datos de entrenamiento en lugar de aprender patrones y características generales que se apliquen a nuevos datos. Como resultado, el rendimiento del modelo puede

ser excelente en los datos de entrenamiento, pero se deteriora en datos no vistos. Para evitar el sobreajuste, se utilizan técnicas como la regularización, la reducción del tamaño de la red neuronal y el uso de conjuntos de validación para supervisar el rendimiento durante el entrenamiento.

Por otro lado, el subajuste, también conocido como infraajuste, ocurre cuando el modelo no logra capturar patrones y características importantes en los datos de entrenamiento. Esto puede ser el resultado de una red neuronal demasiado simple o poco entrenamiento. Como resultado, el rendimiento del modelo puede ser deficiente tanto en los datos de entrenamiento como en los datos de prueba. Para superar el subajuste, se pueden aplicar técnicas como el aumento de datos, el aumento de la complejidad de la red neuronal y el aumento del número de épocas de entrenamiento.

El objetivo es encontrar un punto óptimo en el que el modelo tenga la capacidad de generalizar adecuadamente a nuevos datos sin ajustarse demasiado ni quedarse corto. Esto requiere experimentación y ajuste de los hiperparámetros del modelo, así como el uso de técnicas adecuadas de regularización y validación cruzada [10].

Algunas técnicas y estrategias utilizadas en esta etapa son:

- **Almacenamiento de pesos**: Es recomendable guardar periódicamente los pesos del modelo durante el entrenamiento. Esto permite recuperar el modelo en un estado previo y facilita la reanudación del entrenamiento o la evaluación posterior.
- **Early stopping**: Consiste en monitorear el rendimiento del modelo en un conjunto de validación y detener el entrenamiento si no se observa una mejora significativa durante un cierto número de épocas consecutivas. Esto evita el sobreajuste y permite seleccionar el mejor modelo en función del rendimiento en datos no vistos.
- **Política de actualización del learning rate**: El learning rate o tasa de aprendizaje controla el tamaño de los pasos de actualización de los pesos durante el entrenamiento. Utilizar una política adecuada para ajustar el learning rate a lo largo del entrenamiento puede mejorar la convergencia y la estabilidad del modelo.
- **Regularización**: La regularización es una técnica que introduce términos adicionales en la función de pérdida para penalizar pesos grandes y evitar el sobreajuste. Las formas comunes de regularización incluyen la regresión L1, la regresión L2 y la regularización de dropout.
- **Batch normalization**: Es una técnica que normaliza las activaciones de cada capa durante el entrenamiento, lo que ayuda a estabilizar y acelerar el proceso de aprendizaje. Mejora la capacidad de generalización del modelo y permite utilizar tasas de aprendizaje más altas.
- **Dropout**: Consiste en desactivar aleatoriamente un porcentaje de las neuronas durante el entrenamiento. Esto evita la dependencia excesiva entre neuronas y fomenta la generalización del modelo.
- **Aumento de datos**: Esta técnica, aplicable principalmente en el caso de datos de imágenes, consiste en generar versiones modificadas o aumentadas de los datos de entrenamiento mediante transformaciones como rotaciones, traslaciones, cambios de escala, entre otros. El objetivo es aumentar la diversidad del conjunto de datos y mejorar la capacidad del modelo para generalizar a nuevas muestras.

Estas técnicas de control y seguimiento son fundamentales para obtener modelos más robustos, con un mejor rendimiento y una mayor capacidad de generalización.

3.2.5. Etapa de inferencia

En la etapa de inferencia, se selecciona la red neuronal entrenada que mejor se ajuste a los requisitos del problema. Es importante destacar que el problema abordado en este proyecto es de naturaleza multiclase, lo que significa que implica la clasificación de múltiples categorías o clases. Esta elección de la red puede depender de varios factores, como el rendimiento en conjuntos de validación, la complejidad del modelo y los recursos computacionales disponibles.

Una vez seleccionada la red, se procede a su aplicación en datos nuevos para realizar predicciones en el contexto multiclase mencionado. Durante esta etapa, es crucial evaluar el rendimiento del modelo mediante métricas específicas para problemas multiclase. Las métricas que se explican a continuación se han interpretado desde la perspectiva de un problema multiclase. Estas métricas proporcionan información detallada sobre la calidad de las predicciones y ayudan a evaluar el modelo en el contexto multiclase.

- **Matriz de confusión**: La matriz de confusión es una herramienta fundamental en la evaluación del rendimiento de un modelo de clasificación, especialmente en problemas multiclase, donde se deben considerar varias categorías o clases. Esta matriz se utiliza para visualizar y analizar cómo el modelo ha clasificado las instancias en cada una de las clases disponibles. En la Figura 15 se muestra un ejemplo de matriz de confusión para un problema multiclase de especies.

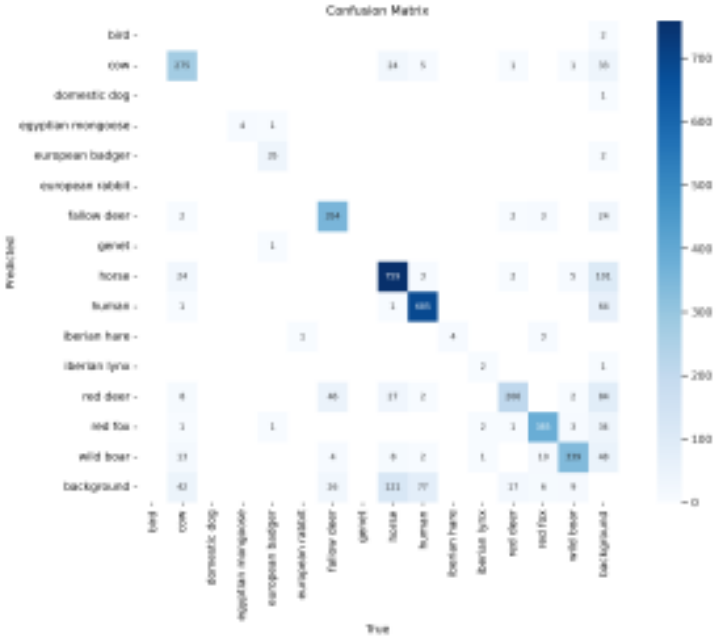

Figura 15: Ejemplo de una matriz de confusión multiclase

Fuente: Elaboración Propia

Donde:

- TP (True Positives): Representa el número de instancias que han sido correctamente clasificadas como pertenecientes a la clase correspondiente. Por ejemplo, si una instancia de la clase A se clasifica correctamente como A, eso sería un verdadero positivo (TP) para la clase A.

- FP (False Positives): Indica el número de instancias que se han clasificado incorrectamente como pertenecientes a una clase en particular, cuando en realidad no lo son. Por ejemplo, si una instancia de la clase B se clasifica incorrectamente como A, eso se consideraría un falso positivo (FP) para la clase A.
- FN (False Negatives): Representa el número de instancias que se han clasificado incorrectamente como no pertenecientes a una clase en particular, cuando en realidad sí pertenecen a esa clase. Por ejemplo, si una instancia de la clase A se clasifica incorrectamente como no perteneciente a la clase A, eso sería un falso negativo (FN) para la clase A.
- TN (True Negatives): Indica el número de instancias que se han clasificado correctamente como no pertenecientes a la clase correspondiente. Por ejemplo, si una instancia que no pertenece a ninguna de las clases A, B, C o D se clasifica correctamente como no perteneciente a esas clases, eso sería un verdadero negativo (TN).

La matriz de confusión proporciona información valiosa sobre el rendimiento del modelo en cada clase, lo que permite calcular diversas métricas de evaluación, como la precisión, la exhaustividad y la puntuación F1, que son fundamentales para comprender la capacidad del modelo para clasificar correctamente cada clase en un problema multiclase.

- ***Accuracy* (Exactitud)**: La exactitud es una métrica que mide la proporción de predicciones correctas, tanto positivas como negativas, realizadas por el modelo respecto al total de instancias en el conjunto de datos. Se calcula utilizando la fórmula:

$$Accuracy = \frac{\text{Número total de predicciones correctas}}{\text{Número total de instancias}}$$

La exactitud es útil para evaluar el rendimiento global del modelo en un problema multiclase, proporcionando una medida general de su capacidad para clasificar correctamente todas las clases. Sin embargo, puede no ser la métrica más adecuada en casos donde las clases están desequilibradas, ya que un modelo podría alcanzar una alta exactitud al predecir siempre la clase mayoritaria.

- ***Precision* (Precisión)**: Mide la proporción de predicciones positivas correctas respecto al total de predicciones positivas realizadas. Se calcula utilizando la fórmula:

$$\text{Precision} = \frac{\text{Verdaderos positivos}}{\text{Verdaderos positivos} + \text{Falsos positivos}}$$

La precisión es útil cuando el objetivo es minimizar los falsos positivos, es decir, las instancias clasificadas incorrectamente como positivas.

- ***Recall* (Exhaustividad)**: Mide la proporción de instancias positivas correctamente identificadas respecto al total de instancias positivas en el conjunto de datos. Se calcula utilizando la fórmula:

$$\text{Exhaustividad} = \frac{\text{Verdaderos positivos}}{\text{Verdaderos positivos} + \text{Falsos negativos}}$$

La exhaustividad es útil cuando el objetivo es minimizar los falsos negativos, es decir, las instancias clasificadas incorrectamente como negativas.

- ***F1 Score* (Puntuación F1)**: Combina la precisión y exhaustividad en una única métrica que proporciona un equilibrio entre ambas. Se calcula utilizando la fórmula:

$$\text{Puntuación F1} = 2 \times \frac{\text{Precisión} \times \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}}$$

  La puntuación F1 es útil cuando se busca un equilibrio entre la precisión y la exhaustividad.

Además, las curvas de rendimiento, como la curva ROC (*Receiver Operating Characteristic*) y la curva de *precision-recall*, ofrecen una representación gráfica del rendimiento del modelo en diferentes umbrales de clasificación. Estas curvas ayudan a comprender el equilibrio entre la tasa de verdaderos positivos y la tasa de falsos positivos, así como la precisión y la exhaustividad del modelo.

- **Curva ROC**: Representa la tasa de verdaderos positivos (sensibilidad) en función de la tasa de falsos positivos (1-especificidad) para diferentes umbrales de clasificación. Permite evaluar el rendimiento del modelo en la capacidad de distinguir entre clases positivas y negativas.
- **Curva de precision-recall**: Representa la precisión en función de la exhaustividad para diferentes umbrales de clasificación. Ayuda a entender el equilibrio entre la precisión y la exhaustividad del modelo en la clasificación de instancias positivas.

En resumen, la etapa de inferencia implica la selección de la mejor red neuronal entrenada y la evaluación de su rendimiento mediante métricas y curvas de rendimiento de clasificación. Esto permite medir la efectividad y la capacidad de generalización del modelo en la tarea específica para la cual fue diseñado.

4. Redes Neuronales Convolucionales

4.1. Introducción a las Redes Neuronales Convolucionales

En el capítulo anterior se presentó una introducción general a las redes neuronales. En esta sección, se profundizará en un tipo específico de redes neuronales llamadas redes neuronales convolucionales, que se basan en el funcionamiento de las redes neuronales y la corteza visual humana.

A diferencia de las redes neuronales convencionales, que están compuestas exclusivamente por capas de neuronas artificiales, las redes convolucionales incorporan otros tipos de capas en su arquitectura para simular el funcionamiento de la corteza visual. Las diferentes capas que pueden aplicarse a las Redes Neuronales Convolucionales (*Convolutional Neural Networks*, CNN) se explican en la Sección 4.2. Estas capas permiten que las redes convolucionales capturen características espaciales y patrones visuales de manera más efectiva. En lugar de procesar los datos de manera global, como lo hacen las redes neuronales tradicionales, las capas convolucionales se enfocan en regiones locales de la entrada y aplican filtros convolucionales para extraer características específicas. Esto les brinda a las redes convolucionales la capacidad de reconocer objetos, formas y estructuras en imágenes y otros datos visuales.

Las CNN son un tipo de arquitectura de redes neuronales especializadas en el procesamiento y análisis de imágenes. Su desarrollo se ha inspirado en los descubrimientos realizados por Hubel y Wiesel en 1959 [37], quienes demostraron que las células en la corteza visual de los animales desempeñan un papel fundamental en la detección de la luz en los campos receptivos. Estos hallazgos sentaron las bases para el surgimiento de las CNN.

En 1980, Kunihiko Fukushima propuso la *neocognition* [38], una arquitectura que se considera el precursor directo de las CNN. Sin embargo, fue en 1990 cuando Lecun et al. publicaron un artículo [39] estableciendo la estructura moderna de una CNN. En dicho trabajo, desarrollaron una red neuronal artificial multi-capas llamada *LeNet-5*, que se destacó por su capacidad para clasificar dígitos manuscritos de manera efectiva. La estructura de esta red se puede ver en la Figura 16.

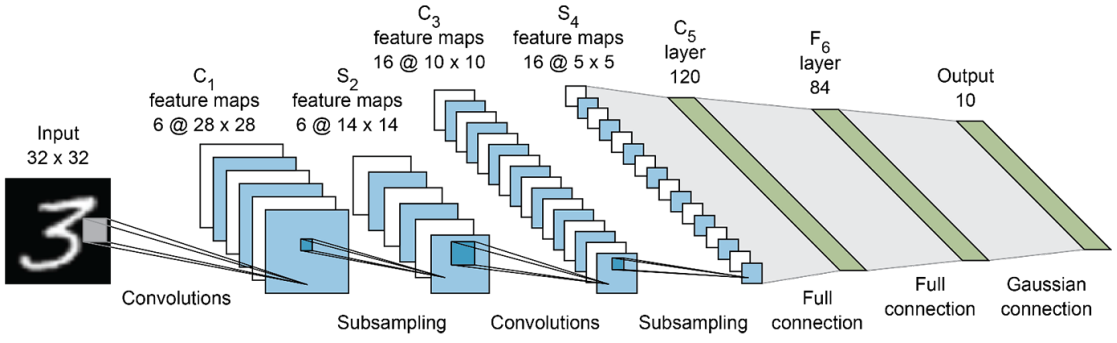

Figura 16: *Estructura de LeNet-5*

Fuente: [39]

Desde entonces, las CNN han experimentado un rápido avance y se han convertido en la metodología dominante en el campo del aprendizaje profundo aplicado a la visión por computador. Estas redes se han utilizado con éxito en diversas tareas, como la clasificación de imágenes, la detección y localización de objetos, el reconocimiento facial, la segmentación semántica y el procesamiento de videos, entre otras aplicaciones [40].

Una de las diferencias clave entre el enfoque de las redes neuronales convencionales frente a las CNN es la entrada de la red. Mientras que en una red neuronal tradicional, un experto debía seleccionar las características relevantes del dominio para alimentar la red, en una red convolucional, se busca encontrar automáticamente estas características a partir de los datos originales.

Este hecho fue demostrado por Matthew D. Zeiler y Rob Fergus en un artículo publicado en 2014, donde mostraron cómo las redes convolucionales tienen la capacidad de reconocer características complejas a mayor profundidad [41].

Por lo tanto, una red convolucional debe encontrar los valores adecuados para cada kernel de convolución presente en la red, así como otros parámetros de diferentes capas, como las conexiones entre unidades en la red neuronal.

La búsqueda de estos valores se basa en una estructura de red predefinida. La elección de la estructura de la red condicionará los resultados, ya que limitará las posibilidades de extraer más información de la imagen de entrada. Será necesario realizar pruebas con diferentes estructuras de red, donde se decida el número y tipo de capas, así como los parámetros de cada capa, como el tamaño del kernel de convolución. La estructura no solo afectará las limitaciones de la red, sino también el número de parámetros que debe aprender. Cuantas más capas haya, más parámetros se deben ajustar. Esto plantea la necesidad de encontrar un equilibrio entre el rendimiento y la eficiencia, ya que un tamaño mayor implica un mayor tiempo de entrenamiento y ejecución.

Cuando hablamos del aprendizaje de una red, nos referimos al ajuste de los valores que reducen el error en la salida de la red. Esto implica decidir cómo se calculará el error y cómo se actualizarán los parámetros en base a este error utilizando un algoritmo de entrenamiento.

El resultado obtenido de la red dependerá de varios factores, como la estructura elegida, la inicialización de los parámetros, el algoritmo de entrenamiento, el tiempo de entrenamiento y el conjunto de entrenamiento utilizado.

La extracción de características se lleva a cabo mediante capas de convolución. Estas capas, combinadas con otras operaciones, permiten detectar características cada vez más complejas a medida que se profundiza en la red. En nuestro ejemplo, las primeras capas pueden buscar características básicas como el color o las líneas, mientras que las capas más profundas pueden detectar la presencia de objetos más complejos, como las ruedas de un automóvil.

En los siguientes apartados, se explorarán las diferentes capas que pueden tener las CNN, así como las arquitecturas más populares y los avances más recientes en este emocionante campo. Se abordarán los tipos de capas utilizados en las CNN, los principios de su funcionamiento y las estrategias de entrenamiento y optimización específicas para este tipo de redes. Además, se presentarán ejemplos de arquitecturas CNN de referencia que han demostrado un rendimiento sobresaliente en diversas tareas de visión por computador.

4.2. Tipos de capas

A continuación, se presentan algunos de los diferentes tipos de capas utilizadas en las arquitecturas de CNN:

- **Capa Convolucional:** La capa convolucional es uno de los tipos de capas más importantes en las redes convolucionales [25]. Está especialmente diseñada para el procesamiento de datos espaciales, como imágenes.

  Se basa en la realización de operaciones de convolución, que implica la aplicación de un filtro, también conocido como kernel de convolución, a la imagen de entrada. El kernel de convolución es una matriz que contiene valores numéricos que actúan como pesos para ponderar la contribución de cada píxel en el cálculo de un nuevo valor en el mapa de características [10].

  Un mapa de características es una representación visual de la información aprendida por la capa convolucional. Se compone de una matriz de valores que destacan la presencia de ciertas características, como bordes, texturas o patrones específicos, en diferentes regiones de la imagen. Cada valor en el mapa de características indica la activación o importancia de una característica particular en una ubicación determinada de la imagen.

  Esta capa realiza una serie de convoluciones sobre la entrada, que puede ser la imagen de entrada o la salida de otra capa de convolución. La invariancia a la translación es una propiedad clave de la convolución, lo que significa que los resultados obtenidos no dependen de la posición del objeto en la imagen [10]. Esto permite que la red pueda reconocer patrones independientemente de su ubicación en la imagen.

  La salida se obtiene mediante la concatenación de las salidas de cada operación de convolución realizada en la capa. Al utilizar una capa de convolución, es necesario definir varios parámetros, como el número de convoluciones, el padding, el stride y el tamaño del kernel. Estos parámetros afectan tanto la dimensión de la salida como el número de parámetros que la red debe aprender [40].

  - **Número de convoluciones**: cantidad de operaciones de convolución que se van a realizar en la capa.
  - ***Padding***: permite añadir elementos adicionales en los límites del mapa de características de entrada para que el kernel de convolución procese los límites de la entrada.
  - ***Stride***: indica sobre qué elementos de la entrada se realiza la operación de convolución, controlando la distancia entre las convoluciones sucesivas.
  - **Tamaño del kernel**: define la dimensión de la matriz de convolución utilizada en la operación de convolución.

  En la Figura 17 se ilustra un ejemplo de convolución de una imagen utilizando un núcleo de convolución de detector de bordes. Esta imagen representa cómo la capa convolucional realiza la operación de convolución entre el núcleo y la imagen, generando un nuevo mapa de características que resalta los bordes presentes en la imagen.

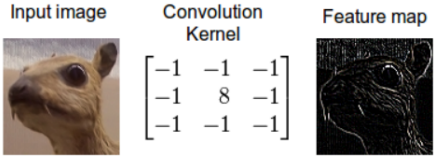

Figura 17: Convolución de una imagen con un núcleo de convolución de detector de bordes

Fuente: https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/convolutional-neural-network

- **Capa de Agrupación (Pooling):** Es una capa que reduce la dimensionalidad de los mapas de características generados por las capas convolucionales. Esta capa combina localmente las características cercanas en una sola representación, reduciendo la cantidad de parámetros y permitiendo la detección de características invariantes a pequeñas traslaciones o deformaciones en los datos.

  La capa de pooling opera sobre cada mapa de características por separado, dividiéndolo en regiones no superpuestas (ventanas de muestra) y aplicando una función de pooling para obtener un valor representativo de cada región. Dos parámetros clave que se utilizan en la capa de pooling son el tamaño de la ventana de muestra y el stride.

  - **Tamaño de la ventana de muestra**: dimensiones de la región cuadrada (o rectangular) que se toma de cada mapa de características para aplicar el pooling. Especifica cuántos píxeles se toman en cuenta en cada dirección (horizontal y vertical). Un tamaño de ventana más pequeño resulta en una reducción más drástica de la dimensionalidad, mientras que un tamaño de ventana más grande preserva más información.
  - **Stride**: determina el desplazamiento de la ventana de muestra a medida que se aplica el pooling. Con un stride de 1, la ventana se desplaza una unidad a la vez, lo que significa que se solapan regiones adyacentes. Con un stride mayor que 1, la ventana se desplaza más rápidamente, lo que resulta en regiones no superpuestas. El stride afecta la tasa de reducción de la dimensionalidad y la cantidad de información preservada.

  Existen varios tipos de pooling utilizados en las redes convolucionales:

  - **Max pooling:** En este tipo de pooling, se toma el valor máximo de cada ventana de muestra. Esto implica seleccionar la característica más destacada en cada región [10].
  - **Average pooling:** En este caso, se calcula el promedio de los valores en cada ventana de muestra. Proporciona una forma de resumir la información general en cada región, sin destacar características específicas [10].
  - **Learned p-norm pooling:** Este tipo de pooling introduce parámetros adicionales para aprender los pesos que se aplican a los valores en cada ventana de muestra. El valor de p determina el grado de normalización. Cuando p = 1, se obtiene una forma de average pooling. Para valores mayores de p, se enfatizan los valores más grandes, mientras que para valores menores de p, se da más importancia a los valores más pequeños [42].

  En la Figura 18 se muestra el resultado de aplicar una operación de max-pooling con tamaño de ventana 2x2 y un stride de 2.

- **Capa Densa (Fully Connected):** Es una capa en la que todas las neuronas están conectadas con todas las neuronas de la capa anterior. Cada conexión tiene un peso asociado que determina la influencia de una neurona en la otra. Esta capa es comúnmente utilizada en redes neuronales feedforward, donde la información fluye en una dirección, de la entrada a la salida.

  La capa densa puede recibir los siguientes parámetros de entrada:

  - **Número de unidades/neuronas:** Define la cantidad de neuronas en la capa densa. Cada neurona en esta capa está conectada con todas las neuronas de la capa anterior.
  - **Función de activación:** Especifica la función matemática utilizada para calcular la salida de cada neurona en la capa densa. Estas funciones fueron ya explicadas en la sección 3.1.2

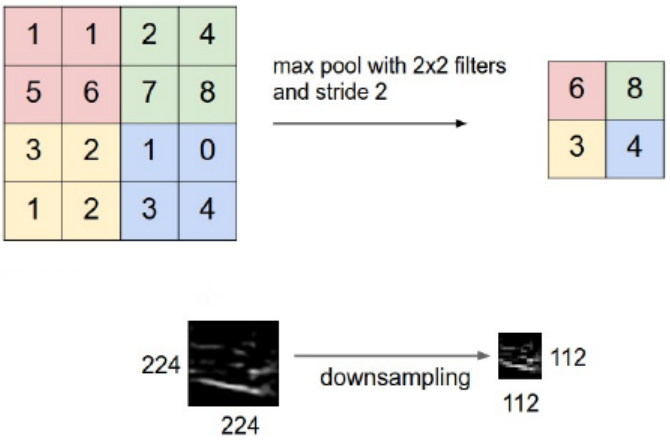

Figura 18: Ejemplo de una operación de *max-pooling*

Fuente: https://cs231n.github.io/convolutional-networks/pool

A continuación, en la Figura 19 se muestra un ejemplo de una red simple formada por capas densas.

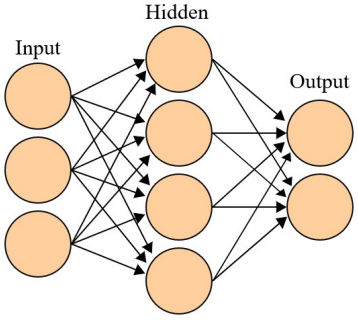

Figura 19: *Capa Densa*

Fuente: https://www.theprofessionalprogrammer.com/2018/11/neural-network-dense-layers.html

- **Capa de Recurrencia (Recurrent):** Son capas capaces de modelar secuencias y dependencias temporales en los datos. Estas capas tienen conexiones recurrentes que permiten que la salida de una unidad se realimente como entrada en la siguiente unidad, lo que les otorga la capacidad de mantener estados internos y recordar información a largo plazo. En la Figura 20 se muestra una red simple con una capa de recurrencia.

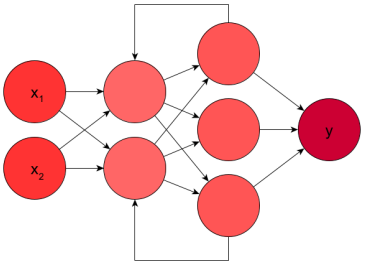

Figura 20: Ejemplo una red con una capa de recurrencia

Fuente: Elaboración propia

- **Capa de Normalización (Normalization):** Son capas que se utilizan para normalizar los valores de las salidas de las capas anteriores. Esto ayuda a mejorar la estabilidad y el rendimiento de la red, al reducir el impacto de los gradientes explosivos o desvanecientes durante el entrenamiento. Algunos ejemplos de capas de normalización son la normalización por lotes (Batch Normalization) y la normalización por capa (Layer Normalization).

- **Capa de upsampling o *transposed convolution*** Es una capa especializada en aumentar el tamaño espacial de los datos.

  Esta capa realiza una operación de interpolación y asigna valores a nuevas ubicaciones para expandir la resolución de los mapas de características.

  Es comúnmente utilizada en tareas de generación de imágenes y en modelos de segmentación semántica para aumentar el tamaño de las representaciones espaciales.

  En la Figura 21 podemos ver el resultado de aplicar una operación de transposed convolution con kernel 3x3 sobre una entrada de tamaño 3x3, con un padding de 1 y un stride de 2, dando como resultado una salida de tamaño 5x5 (normalmente los huecos se rellenan con ceros).

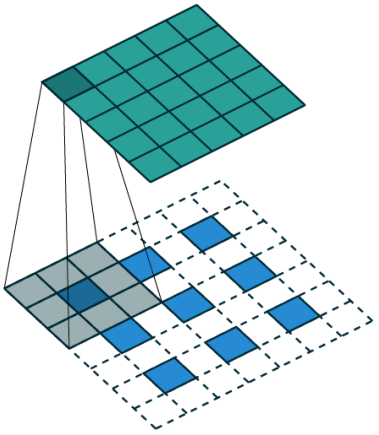

Figura 21: Ejemplo de operación de *transposed convolution*

Fuente: https://github.com/vdumoulin/conv_arithmetic

- **Capa de *Dropout*** Es una capa que consiste en una técnica de regularización comúnmente utilizada en redes neuronales para prevenir el sobreajuste. Durante el entrenamiento, esta capa desactiva aleatoriamente un subconjunto de neuronas, lo que promueve la independencia entre ellas y reduce la dependencia de un conjunto específico de neuronas. Esta introducción de *ruido* ayuda a que el modelo generalice mejor y sea más robusto, ya que no se basa en la activación de un conjunto particular de neuronas. Durante la fase de inferencia, todas las neuronas se activan nuevamente, pero se ajustan multiplicándolas por un factor de escala. La capa de *Dropout* ha demostrado ser efectiva en la reducción del sobreajuste y mejora de la generalización en una variedad de tareas y arquitecturas de redes neuronales [43].

- **Capa de activación:** Es una capa que introduce no linealidad en el modelo. Después de realizar una operación de transformación lineal en los datos de entrada, se aplica una función de activación para introducir la no linealidad en la salida de la capa. Esto permite que la red neuronal aprenda relaciones complejas y represente funciones no lineales. Algunas funciones de activación comunes incluyen la función ReLU (*Rectified Linear Unit*), la función sigmoide, la función tangente hiperbólica o la función *softmax*. Estas y otras funciones de activación fueron explicadas en la sección 3.1.2

Estos son solo algunos ejemplos de los tipos de capas utilizadas en las arquitecturas de redes neuronales. La elección de las capas adecuadas depende del tipo de problema y de las características de los datos a procesar. La combinación y configuración de estas capas en una red neuronal determina la arquitectura específica que se utilizará para resolver un problema particular.

4.3. Arquitecturas populares

En el campo de la clasificación de imágenes, han surgido varias arquitecturas de CNN que han alcanzado un gran rendimiento en tareas de reconocimiento de objetos en el conjunto de datos de ImageNet.
: A continuación, se realiza un repaso general de las arquitecturas más populares de este concurso:

4.3.1. LeNet

La arquitectura LeNet [25] es una de las primeras CNN propuestas. Fue diseñada inicialmente para reconocer caracteres escritos a mano y se hizo famosa por su aplicación en la clasificación de dígitos en el conjunto de datos MNIST. La red consta de 10 clases correspondientes a los dígitos del 0 al 9, y se entrena con 60,000 imágenes en escala de grises de tamaño 32x32.

La arquitectura de LeNet se compone de un total de 7 capas de cuatro tipos diferentes: convolución, *pooling*, activación y densa (*fully connected*). Utiliza filtros de convolución de tamaño 5x5 para extraer características de las imágenes. Después de cada capa de convolución, se aplica un proceso de *pooling*, específicamente un average *pooling*, para reducir la dimensionalidad y conservar las características más importantes.

LeNet utiliza funciones de activación como la sigmoide y la tangente hiperbólica (*tanh*) para introducir no linealidad en la red y permitir la captura de relaciones más complejas entre las características. Finalmente, la clasificación se realiza a través de una red neuronal completamente conectada con tres capas.

Aunque LeNet es relativamente simple en comparación con las arquitecturas más modernas, sentó las bases y fue pionera en el desarrollo de CNN.

En la Figura 22 se muestra la arquitectura de la red.

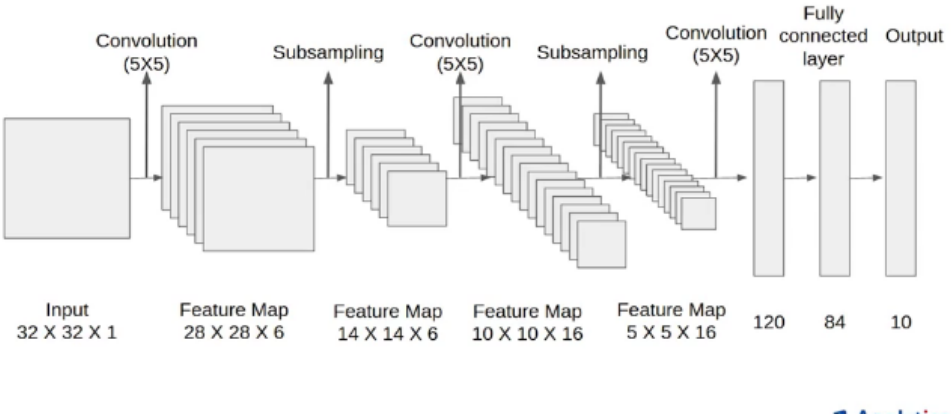

Figura 22: Arquitectura de la red LeNet

Fuente: https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

4.3.2. AlexNet

La arquitectura AlexNet [44] fue la ganadora de la competición ImageNet en 2012 y marcó un hito en la investigación de CNN. AlexNet utiliza capas convolucionales seguidas de capas de *pooling*

y capas densas para la clasificación de imágenes. Esta arquitectura introdujo innovaciones como el uso de la función de activación ReLU y la aplicación de técnicas de regularización, como el *dropout*. AlexNet destacó por su capacidad para aprender características de alto nivel y su rendimiento superior en la clasificación de objetos.

A diferencia de LeNet, AlexNet cuenta con un conjunto de datos mucho más grande, con 1.2 millones de imágenes en color de tamaño 224x224 y 1000 clases para la clasificación. Además, introdujo el uso del *dropout* como una técnica para controlar el sobreajuste durante el entrenamiento. La implementación de AlexNet se realizó en GPUs, lo que permitió acelerar significativamente el entrenamiento, logrando una velocidad 50 veces mayor que con la CPU. La red fue entrenada durante una semana utilizando dos GPUs.

La arquitectura AlexNet ha sido fundamental en el avance del *deep learning*, estableciendo nuevos estándares de rendimiento en tareas de clasificación de imágenes y sentando las bases para el desarrollo de arquitecturas posteriores más complejas y eficientes.

En la Figura 23 se muestra la arquitectura de la red AlexNet, la cual consta de 11 capas con diferentes tipos de convoluciones, además de *pooling* y la función ReLU.

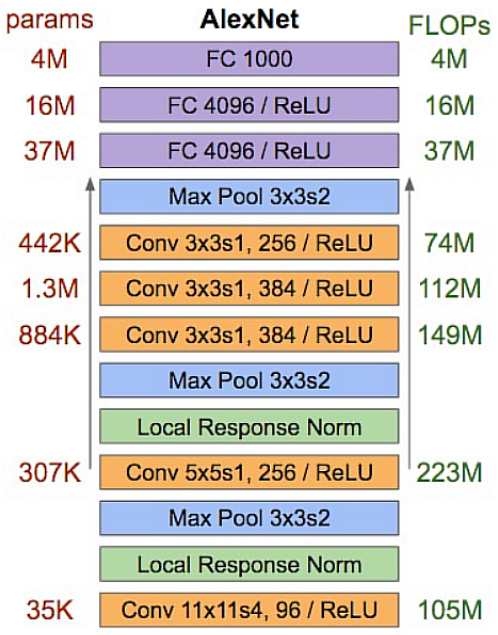

Figura 23: Arquitectura de la red AlexNet

Fuente: https://groups.google.com/forum/!topic/caffe-users/cUD3IF5NMOk

4.3.3. VGG

La arquitectura VGG [45] es conocida por su simplicidad y profundidad. Se caracteriza por utilizar capas convolucionales de 3x3 con un número constante de filtros y capas de pooling intercaladas. La versión más profunda de VGG consta de 19 capas, lo que la convierte en una de las redes neuronales más profundas en su momento. Aunque VGG es más lenta de entrenar y requiere más recursos computacionales debido a su profundidad, logró obtener un rendimiento destacado en

diversas tareas de clasificación de imágenes.

En la Figura 24 se muestra la arquitectura de la red VGG, la cual se compone únicamente de convoluciones de 3x3 con función de activación ReLU entre ellas, junto con capas de max pooling. Esta arquitectura quedó en segundo lugar en la competición ImageNet en 2014, logrando reducir el error al 7.3 %. VGG se diseñó de manera simple pero con un aumento notable en la profundidad de la red, alcanzando un total de 19 capas.

VGG demostró que incluso con una estructura no muy compleja pero lo suficientemente profunda, se pueden obtener buenos resultados en tareas de clasificación de imágenes. Fue en este año cuando se comenzó a considerar el aumento en la profundidad de las redes como una estrategia para mejorar el rendimiento.

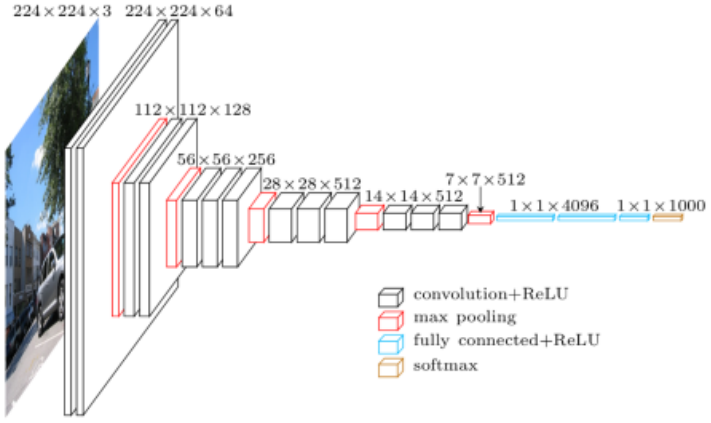

Figura 24: Arquitectura de la red VGG

Fuente: https://paperswithcode.com/method/vgg

4.3.4. GoogleNet (Inception V1)

GoogleNet, también conocida como Inception V1, fue la arquitectura ganadora del desafío ImageNet en 2014 [46]. Esta arquitectura revolucionaria introdujo el concepto de *módulo Inception*, que permitía realizar convoluciones de diferentes tamaños y concatenar sus salidas, permitiendo que la red aprendiera representaciones más ricas a diferentes escalas, se presenta la representación de este módulo en la Figura 25. Esto condujo a una reducción significativa en el número de parámetros que la red debía aprender.

Además, GoogleNet se caracterizó por su eficiencia, logrando un alto rendimiento con un número menor de parámetros en comparación con otras arquitecturas. Para reducir aún más el número de parámetros, GoogleNet utilizó el promedio de pooling en lugar de una capa fully connected al final de la red. La arquitectura constaba de un total de 100 capas y su rendimiento destacado en la competición ImageNet impulsó el desarrollo de redes más profundas.

En la Figura 26 se muestra la arquitectura de la red GoogleNet. Esta arquitectura no seguía una estructura secuencial convencional, gracias a los módulos Inception. Esta innovación sentó las bases para el desarrollo de nuevos modelos basados en los módulos Inception en los años siguientes.

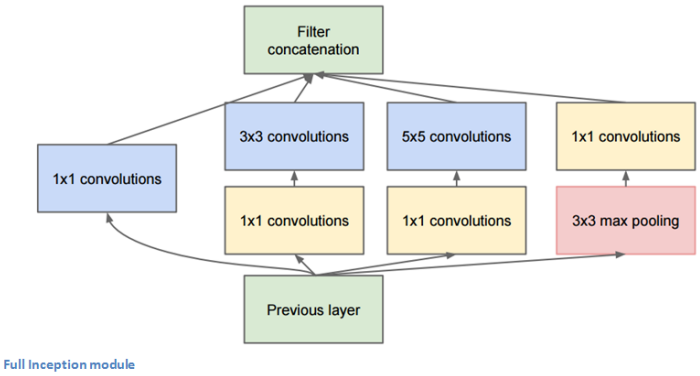

Figura 25: Módulo Inception

Fuente: https://adeshpande3.github.io/assets/GoogLeNet3.png

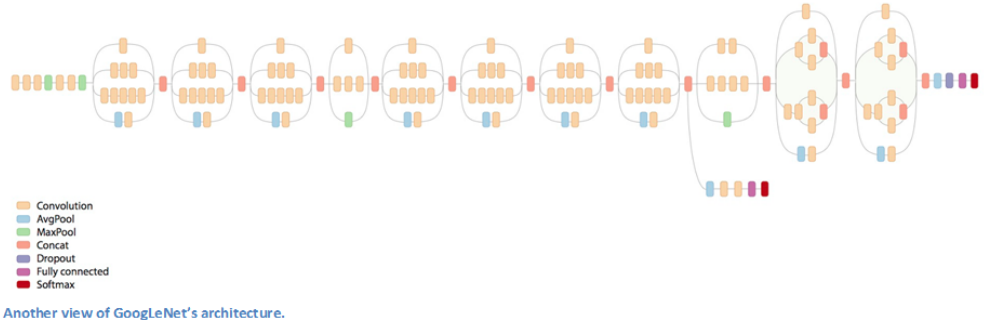

Figura 26: Arquitectura de la red GoogleNet

Fuente: https://adeshpande3.github.io/assets/GoogLeNet.png

4.3.5. ResNet

ResNet, acortamiento de *Residual Network*, es una arquitectura revolucionaria que se introdujo en 2015 para abordar el problema del decaimiento del gradiente en redes neuronales profundas [47]. Su innovación clave es el uso de *conexiones residuales* que permiten que el gradiente fluya directamente a través de las capas sin disminuir su magnitud. En lugar de confiar únicamente en la propagación hacia adelante, ResNet suma la salida de una capa a la entrada original, lo que facilita el entrenamiento de redes neuronales extremadamente profundas.

La arquitectura de ResNet se destaca por su capacidad para entrenar redes con hasta 152 capas (ResNet-152) sin degradación significativa del rendimiento. Esto fue un gran avance en comparación con arquitecturas anteriores que sufrían problemas de rendimiento al aumentar la profundidad. La figura 27 muestra la arquitectura de la red ResNet.

A medida que la información fluye a través de las capas, las conexiones residuales permiten que los gradientes se propaguen sin disminuir, lo que facilita el entrenamiento de redes más profundas. Este enfoque ha demostrado ser altamente efectivo en una variedad de tareas de visión por computador y ha sido ampliamente adoptado en la comunidad de aprendizaje profundo.

4.3.6. ViT-G/14

ViT-G/14, abreviatura de *Vision Transformer - Base, 14 capas* [48], es una arquitectura de Transformer diseñada específicamente para la clasificación de imágenes. A diferencia de las arqui-

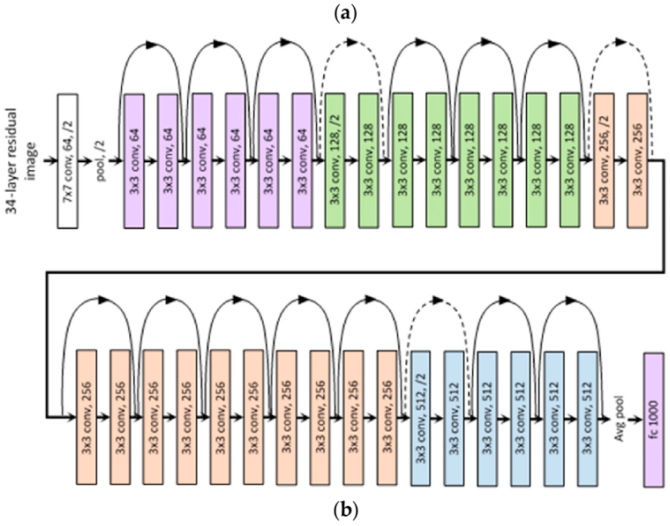

Figura 27: Arquitectura de la red ResNet

Fuente: https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d

tecturas convolucionales tradicionales, ViT utiliza atención y operaciones de transformación para capturar las relaciones entre las diferentes partes de la imagen. Introduce la idea de descomponer la imagen en parches y tratarlos como secuencias de entrada.

La principal característica de ViT es su capacidad para aprender de manera global, utilizando la atención para capturar las dependencias a largo plazo en la imagen. Además, utiliza una codificación de posición para preservar la información espacial de los parches. ViT se ha destacado en la clasificación de imágenes a gran escala y ha mostrado un rendimiento comparable e incluso superior a las arquitecturas convolucionales en varios conjuntos de datos. La figura 28 muestra la arquitectura de la red ViT-G/14.

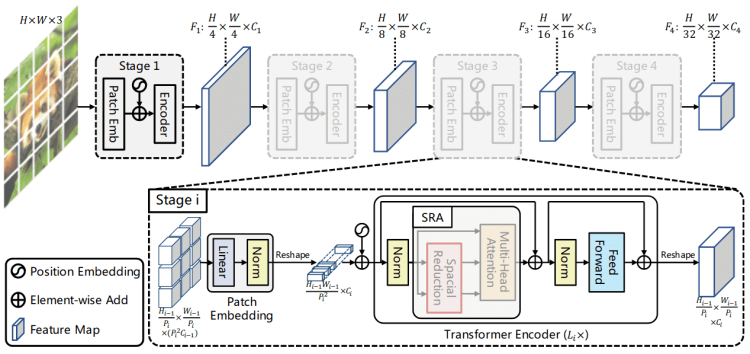

Figura 28: Arquitectura de la red ViT

Fuente: https://theaisummer.com/transformers-computer-vision/

La introducción de Transformers en el campo de la visión por computador ha abierto nuevas posibilidades y ha demostrado ser efectiva en tareas de clasificación de imágenes. ViT ha sido ampliamente adoptada y ha impulsado la investigación en la aplicación de modelos Transformer en otros dominios más allá del procesamiento de lenguaje natural.

4.3.7. CoCa (finetuned)

CoCa (Contrastive Captioner) [27] es un modelo de base que utiliza un enfoque minimalista para el preentrenamiento de un codificador-decodificador de imágenes y texto, combinando la pérdida contrastiva y la pérdida de generación de subtítulos. CoCa omite la atención cruzada en las primeras capas del decodificador para codificar representaciones de texto unimodal y utiliza capas posteriores del decodificador que se atienden cruzadamente con el codificador de imágenes para obtener representaciones multimodales de imágenes y texto. Además, aplica una pérdida contrastiva entre las incrustaciones de imágenes y texto unimodales, junto con una pérdida de subtítulos en las salidas del decodificador multimodal que predice tokens de texto de forma autoregresiva. CoCa se entrena de forma eficiente con un mínimo sobrecoste y se preentrena desde cero en grandes conjuntos de datos de texto alternativos y de imágenes anotadas. Experimentalmente, CoCa logra un rendimiento sobresaliente en una amplia gama de tareas, incluyendo reconocimiento visual, recuperación multimodal y generación de subtítulos de imágenes, superando el estado del arte en clasificación de imágenes con una precisión de top-1 del 91.0 % en ImageNet con un codificador afinado. En la Figura 29 se muestra la arquitectura de la red.

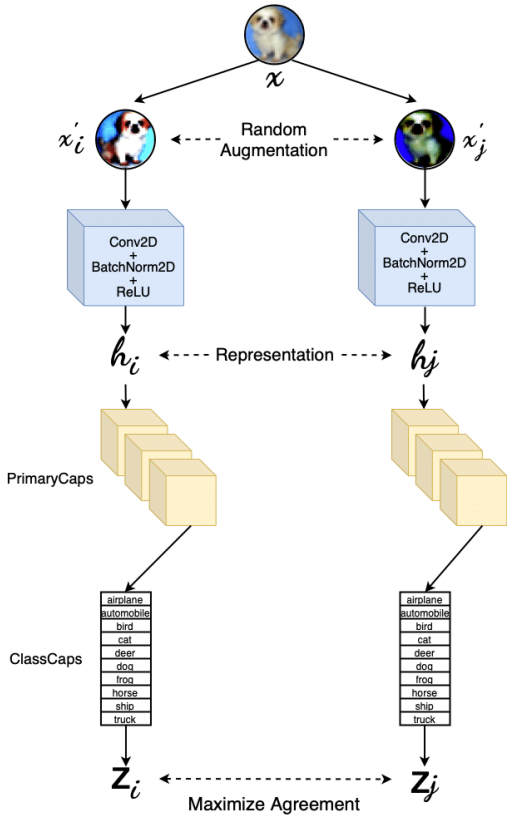

Figura 29: Arquitectura de la red CoCa

Fuente: https://www.researchgate.net/figure/Main-architecture-of-CoCa-proposed-model_fig3_363843373

Estas arquitecturas representan algunos de los grandes avances en el campo de la clasificación de imágenes utilizando redes neuronales. Cada arquitectura tiene sus puntos fuertes y débiles, y su elección depende del problema específico y los recursos disponibles. Es importante tener en cuenta que la investigación y el desarrollo en este campo está en constante evolución, y es posible que haya surgido nuevas arquitecturas desde la redacción de este texto.

4.4. Arquitecturas utilizadas en este trabajo

4.4.1. MobileNetV2

MobileNetV2 [49] es una arquitectura de red neuronal diseñada para aplicaciones de visión por computador en dispositivos móviles y con recursos computacionales limitados. Fue desarrollada como una mejora de la arquitectura original MobileNetV1, con el objetivo de lograr un equilibrio entre la precisión de la red y la eficiencia computacional.

Una de las características que diferencian a MobileNetV2 de otras redes, es el uso de unos bloques de construcción llamados *inverted residual blocks* (bloques residuales invertidos). Estos bloques están compuestos por una combinación de capas convolucionales 1x1 y 3x3, seguidas de una capa convolucional 1x1 lineal. La utilización de estas capas convolucionales reduce la cantidad de cálculos necesarios y permite mantener una representación rica de características.

Además, MobileNetV2 incorpora una técnica llamada *bottleneck* (cuello de botella), que reduce aún más la cantidad de cálculos necesarios. En los bloques residuales invertidos, se utiliza una capa convolucional 1x1 para reducir la dimensionalidad de los datos de entrada, lo que disminuye el número de canales y, por lo tanto, la carga computacional.

Otra característica importante de MobileNetV2 es el uso de *inverted residuals with linear bottleneck* (bloques residuales invertidos con cuello de botella lineal). En lugar de utilizar funciones de activación no lineales, se aplica una función de activación lineal en las capas convolucionales 1x1 intermedias, lo que permite que la red capture representaciones más lineales y evita la introducción de no linealidades innecesarias.

En la Figura 30 se puede observar la distribución de las capas en la arquitectura que se está describiendo.

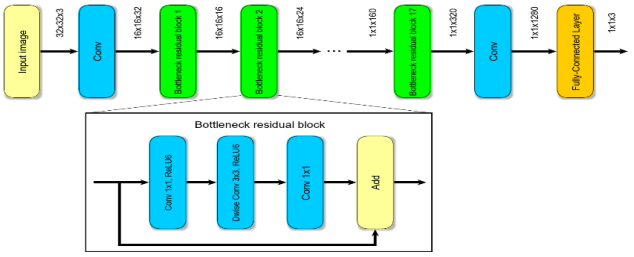

Figura 30: Arquitectura de la red MobileNetV2

Fuente: https://www.researchgate.net/figure/The-architecture-of-the-MobileNetv2-network_fig3_342856036

En resumen, MobileNetV2 es una arquitectura de red neuronal diseñada para ofrecer un equilibrio óptimo entre precisión y eficiencia computacional en dispositivos móviles. Su diseño incluye bloques residuales invertidos, uso de capas convolucionales 1x1 y 3x3, cuellos de botella lineales y otras técnicas para reducir la carga computacional sin comprometer significativamente el rendimiento. Ha demostrado ser eficaz en diversas tareas de visión por computador y ha sido ampliamente adoptada en aplicaciones móviles y embebidas.

4.4.2. *Arquitectura de Crohn*

La arquitectura que en este documento trataremos como *Arquitectura de Crohn* es la propuesta en el trabajo *Automatic detection of Crohn disease in wireless capsule endoscopic images using a deep convolutional neural network* [50]. Esta arquitectura fue diseñada para abordar el problema de clasificación de imágenes capturadas por endoscopias de cápsula inalámbrica (CE) para identificar lesiones indicativas de la enfermedad de Crohn.

La motivación para utilizar la *Arquitectura de Crohn* radica en su naturaleza más reducida y especializada en la clasificación de imágenes, mientras que la otra arquitectura propuesta es más compleja. Al emplear la *Arquitectura de Crohn*, podremos contrastar los resultados con una arquitectura más simplificada y evaluar cómo se desempeña en comparación con MobileNetV2. Dado que la *Arquitectura de Crohn* fue diseñada específicamente para detectar la enfermedad de Crohn en imágenes de intestino delgado capturadas mediante cápsula ensoscópica, o videocápsula, es posible que muestre un rendimiento superior en esta tarea. Esto resalta la importancia de una arquitectura adaptada al problema en cuestión. Además, al comparar ambas arquitecturas, podremos identificar sus beneficios y limitaciones, lo que nos permitirá comprender mejor cómo la elección de la arquitectura puede influir en los resultados de la clasificación, teniendo en cuenta la relevancia del *dataset* y su adecuación al problema específico a resolver.

La arquitectura de CNN propuesta se centra en la clasificación de imágenes del intestino delgado en dos tipos: patológicas y sanas, en función de la presencia o ausencia de lesiones relacionadas con la enfermedad de Crohn. El diseño de la arquitectura se adaptó específicamente a este problema de clasificación de imágenes, tomando decisiones de diseño con el objetivo de mejorar el rendimiento en términos de precisión y velocidad de procesamiento en comparación con otras arquitecturas de referencia basadas en aprendizaje profundo.

La Figura 31 muestra la distribución de las capas en la *Arquitectura de Crohn*. La arquitectura consta de 6 bloques dedicados a la extracción de características, y cada bloque tiene una estructura similar. En cada bloque, se realiza una operación de convolución en el tensor de entrada con un kernel de 3x3, un paso de 1 y un relleno de 1. A continuación, se aplica una normalización por lotes para acelerar y facilitar la convergencia del entrenamiento, seguida de una operación de ReLU para introducir no linealidades en el modelo. Los bloques difieren únicamente en el número de capas de convolución incluidas, que aumentan progresivamente de 32 a 96.

La arquitectura también incluye capas de agrupamiento (pooling) para reducir el tamaño de los datos generados en la salida de cada bloque. Después de los primeros 5 bloques, se aplica una operación de maxpooling con un tamaño de ventana de 3x3, un paso de 2 y un relleno de 1. En la salida del último bloque, se aplica un agrupamiento promedio global. La arquitectura se completa con una capa de red neuronal con dos neuronas, asociadas a cada una de las dos clases del problema: imágenes sanas e imágenes con lesiones. Durante la fase de inferencia, se aplica la función softmax a la salida de la red, generando la probabilidad de pertenencia a cada clase para una imagen de entrada dada.

Es importante destacar que esta arquitectura implementa una estrategia de procesamiento de información notablemente diferente a la utilizada por las principales CNN del estado actual. Estas últimas reducen agresivamente la resolución de la imagen en las primeras capas y aplican un número considerablemente mayor de convoluciones. Sin embargo, la arquitectura propuesta en este trabajo se adapta a las características específicas del problema de clasificación de imágenes endoscópicas,

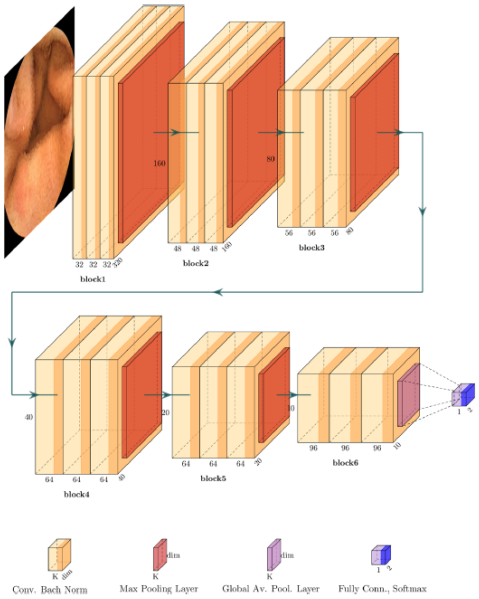

Figura 31: Arquitectura de la *Arquitectura de Crohn*

Fuente: [50]

donde las condiciones son diferentes y la búsqueda se centra en una arquitectura más eficiente en términos de parámetros y velocidad de procesamiento.

En resumen, la arquitectura propuesta en este trabajo presenta una configuración específica para la clasificación de imágenes endoscópicas de la enfermedad de Crohn. Su diseño se adaptó a las necesidades del problema, manteniendo la resolución de las imágenes y utilizando convoluciones de kernel pequeño para reducir el número de parámetros. La arquitectura logró resultados superiores a otras arquitecturas de referencia, tanto en términos de rendimiento como de velocidad de procesamiento, lo que la hace prometedora para su uso en el diagnóstico de la enfermedad de Crohn.

En la experimentación, esta arquitectura se adaptará para tratar dos problemas de clasificación multiclase ya que su diseño fue propuesto para aplicarse en un problema de clasificación binaria.

5. Introducción al problema. Estado del arte

En este capítulo, se aborda el problema propuesto de clasificación de especies de animales en imágenes de fototrampeo mediante el uso de CNN. Se explicará la motivación de este problema y se realizará un breve estado del arte para comprender cómo se ha abordado anteriormente.

5.1. Motivación

La aplicación de técnicas de visión por computador en el campo de la ecología ha despertado un creciente interés debido a las numerosas ventajas que ofrece en la conservación y estudio de la biodiversidad. Una de las principales motivaciones detrás de la realización de estudios en esta área radica en la necesidad de mejorar la eficiencia de los procesos de clasificación de especies. Tradicionalmente, la identificación de especies en entornos naturales ha sido un trabajo laborioso y costoso que requiere la experiencia de personas del ámbito de la biología y ecología. Se realiza mediante salidas de campo y observación directa. Así, este método, además de ser laborioso y costoso, puede estar sujeto a errores humanos y sesgos [51]. La aplicación de técnicas de visión por computador permite automatizar gran parte de este proceso, acelerando la recopilación de datos y proporcionando resultados más precisos y consistentes.

Un primer paso para conseguir la automatización de este proceso puede ser el uso de cámaras de fototrampeo. Estas cámaras se colocan en el hábitat de interés y, cuando los sensores detectan movimiento, capturan imágenes de los animales que pasan frente a ellas. Esto permite recolectar imágenes de forma continua y eficiente, reduciendo la necesidad de salidas de campo frecuentes [52]. Así se puede obtener de forma más automatizada una gran cantidad de imágenes de la biodiversidad del área de estudio. Además, es importante destacar que las técnicas tradicionales de monitoreo de la biodiversidad, pueden incluir el uso de trampas y marcadores. Estas técnicas pueden ser intrusivas y peligrosas tanto para las especies como para los seres humanos. Por eso, el cambio de estas técnicas intrusivas y estresantes para los animales es muy importante, para conseguir un monitoreo más seguro y respetuoso con el medio.

La verdadera utilidad de las cámaras de fototrampeo se alcanza cuando se combinan con técnicas de inteligencia artificial, como las CNN. La clasificación y conteo de individuos en imágenes por parte de personal experto puede resultar muy costoso en muchos aspectos, entre otros, en términos de tiempo. Además de ser una tarea repetitiva y poco atractiva para las personas. Con el desarrollo de sistemas de clasificación de especies basados en *deep learning*, las imágenes recolectadas por las cámaras podrían analizarse y clasificarse automáticamente, permitiendo el conteo y la identificación de animales de manera rápida, precisa y de una manera más económica [53]. Así, el tiempo que las personas invertirían en clasificar y contar los individuos de las imágenes podría ser invertido en otras tareas, como la elaboración de los modelos poblacionales.

En resumen, la automatización permitida por las técnicas de visión por computador en ecología no solo mejora la eficiencia en la clasificación de especies y la creación de modelos poblacionales precisos, sino que también reduce la intrusividad y el riesgo asociado con las técnicas tradicionales de monitoreo, lo que contribuye significativamente a la conservación de la biodiversidad.

5.2. Estado del arte

El conteo manual de animales ha sido una práctica tradicional utilizada en ecología y conservación para estimar la abundancia y distribución de especies en diferentes hábitats. Esta metodología implica que los investigadores realicen salidas de campo para observar y contar directamente los animales presentes en un área de estudio. Si bien este enfoque ha proporcionado datos valiosos a lo largo de los años, presenta limitaciones significativas, como la necesidad de recursos humanos y financieros considerables, así como la posible presencia de sesgos y errores en el conteo.

El uso de cámaras de fototrampeo surgió como una alternativa prometedora al conteo manual. Estas cámaras, equipadas con sensores de movimiento y activadas por el paso de los animales, permiten la captura automática de imágenes sin la intervención directa de los investigadores. El uso de fototrampas ofrece varias ventajas, como una mayor eficiencia en la recopilación de datos, la reducción del disturbio a la fauna y la capacidad de monitorear áreas remotas y de difícil acceso de manera continua [51].

El desarrollo de técnicas de detección de movimiento y sistemas de cámara más sofisticados ha mejorado significativamente la eficiencia y calidad de las imágenes de fototrampeo. Los primeros estudios sobre el uso de cámaras de fototrampeo se centraron en la detección de presencia o ausencia de las especies objetivo a partir de las imágenes capturadas. Estos enfoques iniciales utilizaban métodos tradicionales de visión por computador y análisis de imágenes para identificar la presencia de animales en las fotos [54].

Sin embargo, estos primeros métodos presentaban desafíos en la clasificación y conteo preciso de especies en las imágenes. La variabilidad en la apariencia de los animales, las condiciones de iluminación y el fondo del hábitat dificultaban el desarrollo de algoritmos robustos para la identificación de especies. A medida que la tecnología avanzaba, comenzaron a utilizarse técnicas de aprendizaje automático, como máquinas de soporte vectorial (SVM) y clasificadores basados en descriptores de características locales, para mejorar la precisión en la identificación de especies en imágenes de fototrampeo [55].

El enfoque hacia el uso de técnicas de *deep learning* y CNN en la clasificación de imágenes de fototrampeo marcó un gran avance en la automatización del proceso. Con la capacidad de aprender automáticamente características relevantes de las imágenes, las CNN demostraron una alta precisión en la clasificación de especies [53]. Estos modelos pudieron distinguir patrones y características específicas de cada especie, superando las limitaciones de los enfoques anteriores que requerían una ingeniería manual de características.

La utilización de grandes conjuntos de datos etiquetados y técnicas de transferencia de aprendizaje también ha contribuido a mejorar la precisión de los modelos de clasificación. Al entrenar las CNN en datos de fototrampeo previamente etiquetados, los modelos pueden generalizar y reconocer características comunes a diversas especies, lo que permite una clasificación más precisa de animales en nuevas imágenes [56].

El desarrollo continuo de algoritmos de *deep learning* ha llevado a mejoras adicionales en la clasificación de imágenes de fototrampeo. Por ejemplo, la utilización de arquitecturas avanzadas, como redes neuronales residuales (ResNet) y redes neuronales con atención, ha permitido aumentar aún más la precisión y robustez de los modelos [47]. Además, se han explorado técnicas de aprendizaje semi-supervisado y de etiquetado débil para abordar el desafío de la escasez de datos etiquetados

en la clasificación de especies en imágenes de fototrampeo [53].

A medida que la tecnología de *deep learning* continúa avanzando, se prevé que el análisis de imágenes de fototrampeo y la clasificación de especies seguirán mejorando en términos de precisión y eficiencia. La combinación de cámaras de fototrampeo con técnicas de *deep learning* ha demostrado ser una herramienta poderosa para el monitoreo de la biodiversidad y la conservación de especies, permitiendo una mayor comprensión de los ecosistemas y una toma de decisiones más informada [53]. Sin embargo, también es importante abordar los desafíos éticos y de privacidad asociados con el uso de estas tecnologías, así como seguir mejorando la generalización de los modelos para abarcar una amplia variedad de hábitats y especies [56].

6. Materiales

En esta sección, se presentan los materiales que fueron empleados en el desarrollo de este estudio. Los materiales incluyen los recursos y conjuntos de datos específicos que desempeñaron un papel crucial en la experimentación y evaluación de los modelos. Además, se destacará la relevancia de cada uno de estos elementos en relación con los objetivos y resultados del estudio.

6.1. Base de Datos

El conjunto de datos empleado en este estudio proviene de la base de datos de imágenes de fototrampeo desarrollada en el marco del proyecto AI-Census de la Universidad de Huelva [57]. Esta base de datos se compone de imágenes capturadas por cámaras de fototrampeo desplegadas en diversas ubicaciones desde octubre de 2020 hasta la fecha. Inicialmente, el proyecto comenzó con aproximadamente 30 cámaras trampa y ha ampliado su alcance a la actualidad, abarcando un total de 60 cámaras distribuidas en 60 ubicaciones distintas.

La tarea de etiquetado de las imágenes se ha llevado a cabo principalmente a través de la colaboración de la comunidad en la plataforma Zooniverse [58]. Usuarios voluntarios han participado en el etiquetado de animales y especies presentes en las imágenes. Este etiquetado ciudadano ha generado un valor de confianza asociado a cada etiqueta, basado en la coincidencia de respuestas entre los usuarios. Además, parte significativa de las imágenes ha sido validada por expertos pertenecientes al equipo de investigación para garantizar la precisión de las etiquetas.

El conjunto de datos resultante comprende alrededor de 3.500.000 imágenes etiquetadas, todas capturadas en 60 ubicaciones diferentes y pertenecientes a un total de 21 clases de especies distintas (y 23 clases en total, incluyendo categorías no relacionadas con especies). Sin embargo, se seleccionaron aquellas imágenes cuyas etiquetas superaron un umbral de confianza específico.

El umbral de confianza ha sido determinado considerando la calidad de las etiquetas de las imágenes. Estas etiquetas pueden provenir tanto de expertos en el campo como de la clasificación realizada por la comunidad de usuarios en la plataforma Zooniverse. Para evaluar la confianza en las etiquetas de Zooniverse, se ha creado una métrica artificial que tiene en cuenta cuántos usuarios coincidieron en la clasificación de una imagen. Esta métrica varía entre 0 y 1, donde un valor más cercano a 1 indica un alto grado de consenso entre los usuarios clasificadores.

Después de aplicar un umbral de confianza sobre el conjunto de datos total se dispone de un conjunto de 700.000 imágenes etiquetadas con alta confianza. Es importante destacar que algunas de estas clases son minoritarias y cuentan con menos de 50 imágenes, lo que plantea un desafío adicional para la clasificación de esas imágenes.

Por otro lado, la clase *vacía* es la que contiene el mayor número de imágenes, lo que acentúa el problema del desequilibrio de datos. La distribución del número de imágenes por cada clase en el conjunto de datos se muestra en la Figura 32. Una imagen que pertenece a la clase *vacía* es aquella en la que no hay ningún animal, sólo se ve el paisaje del fondo de la imagen.

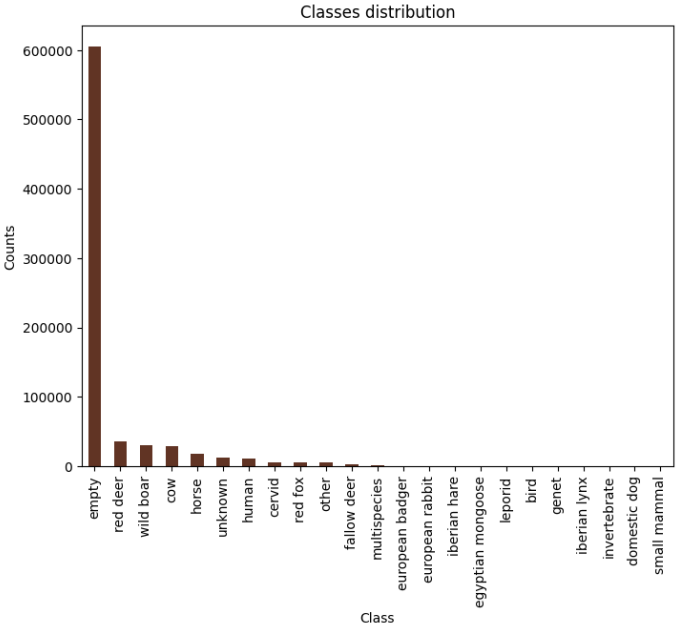

Figura 32: Distribución por clases del *Dataset*

Fuente: Elaboración propia

El objetivo principal de este estudio es la clasificación de animales, y dado que la clase *vacías* contiene una cantidad significativa de imágenes, esta distorsiona la visualización de la distribución de las demás clases. La Figura 33 presenta la distribución del número de imágenes por cada clase del conjunto de datos, excluyendo la clase *vacías*.

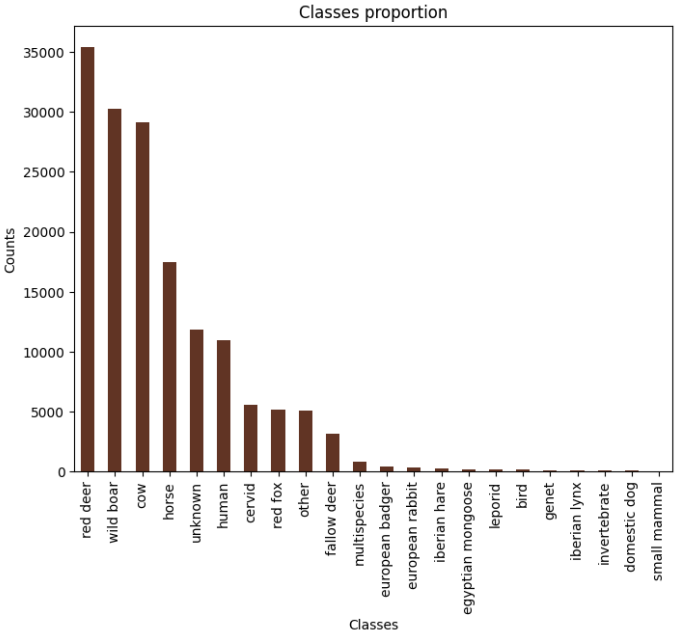

Figura 33: Distribución por clases del *Dataset* sin la clase *empty*

Fuente: Elaboración propia

La utilización de esta base de datos de imágenes de fototrampeo etiquetadas brinda una valiosa

oportunidad para entrenar y evaluar modelos de aprendizaje profundo en la tarea de clasificación automática de especies. La diversidad de especies representadas y las 60 localizaciones diferentes reflejan escenarios realistas y complejos, lo que permite a los modelos generalizar y adaptarse de manera efectiva a una amplia variedad de hábitats y condiciones ambientales.

Además de las imágenes en sí, se dispone de un archivo CSV que contiene información relevante sobre cada imagen, incluyendo su ruta relativa, la clase a la que pertenece, la ubicación en la que fue capturada y la fecha y hora de captura. Esta información adicional facilita la gestión y análisis de los datos, lo que resulta esencial para llevar a cabo un estudio integral sobre la clasificación de especies en el fototrampeo.

6.1.1. Conjunto de Datos

El conjunto de datos utilizado en este estudio ha sido extraído de la base de datos de imágenes de fototrampeo del proyecto AI-Census de la Universidad de Huelva explicado en el apartado anterior 6.1. Para garantizar la eficiencia computacional y enfocar el análisis en las clases más representativas, se han seleccionado únicamente las clases mayoritarias, es decir, aquellas que contenían al menos 1000 imágenes etiquetadas.

En total, se han incluido 8 clases de animales diferentes en el conjunto de datos. Estas clases son: caballo, ciervo, gamo, humano, jabalí, vaca, vacía y zorro. Cada una de estas clases ha sido representada por exactamente 1000 imágenes, lo que proporciona una distribución equitativa y balanceada para el entrenamiento de los modelos de *deep learning*.

El proceso de extracción de las 1000 imágenes de cada clase se llevó a cabo con el objetivo de construir un *dataset* lo más diverso posible, ya que la diversidad es esencial para evitar el sobreaprendizaje y mejorar la generalización del modelo. Se buscó incluir una mezcla equilibrada de imágenes nocturnas y diurnas dentro de cada clase, así como diferentes localizaciones y distancias con respecto a los animales entre otros factores. Sin embargo, debido a la limitada información disponible, no fue posible lograr esta diversidad de manera automática. En lugar de ello, se optó por extraer una muestra uniforme del *dataset* original y total. Es importante tener en cuenta que algunas especies son exclusivamente nocturnas, lo que puede resultar en un número significativamente menor de imágenes diurnas para esas clases, y viceversa. Lo mismo ocurre con las localizaciones, donde algunas especies tienen movimientos restringidos a áreas específicas. Esta situación representa un desafío adicional, ya que dificulta la obtención de la variabilidad deseada en el conjunto de datos extraído. A pesar de estas dificultades, se espera que el *dataset* resultante sea lo suficientemente representativo y adecuado para el entrenamiento y evaluación de modelos de clasificación mediante técnicas de *deep learning*.

La Figura 34 muestra una imagen representativa por cada una de las 8 clases del conjunto de entrenamiento. Estas imágenes ilustran la diversidad de especies y escenarios que se encuentran en la base de datos de fototrampeo, lo que refleja la complejidad y el realismo de los datos utilizados en este estudio.

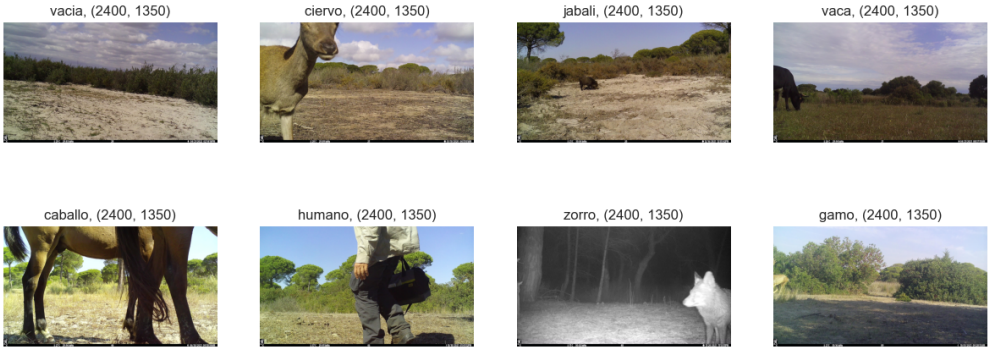

Figura 34: Muestra de imágenes del *Dataset*

Fuente: Elaboración propia

Como se puede observar, el conjunto de datos presenta una amplia variedad de condiciones en las imágenes de fototrampeo. Se pueden encontrar animales en diferentes distancias, algunos muy alejados, otros a media distancia o muy cerca de la cámara. Además, las condiciones lumínicas varían significativamente, con imágenes capturadas tanto durante la noche como durante el día. Otro factor a considerar es la variabilidad en los fondos de las imágenes, que van desde escenarios con vegetación densa hasta áreas abiertas. Estas diferencias en las condiciones de las imágenes serán de gran relevancia, ya que pueden tener una gran repercusión en los resultados del modelo de clasificación.

6.2. Conjuntos de Entrenamiento, Validación y Test

La selección de las imágenes para los conjuntos de entrenamiento, validación y test se realizó siguiendo una proporción de 80 %, 10 % y 10 %, respectivamente, del total de 1000 imágenes por clase extraídas anteriormente. Esto significa que el 80 % de las imágenes serán utilizadas para entrenar los modelos, mientras que el 10 % se reservará para la validación y el 10 % restante para el conjunto de *test* utilizado para medir el rendimiento final del modelo.

La separación de los conjuntos se llevó a cabo de manera uniforme, ya que actualmente no existe un método específico para lograr una distribución balanceada de los datos en este contexto. Aunque se ha intentado obtener una distribución diversa y equilibrada de las imágenes en los conjuntos, las limitaciones mencionadas previamente, como la disponibilidad de información y la naturaleza de ciertas especies, pueden haber influido en la representatividad exacta de cada conjunto. Sin embargo, se espera que esta división proporcione un conjunto de datos adecuado para entrenar y evaluar los modelos de clasificación de especies.

La distribución de las clases se muestra en la Figura 35:

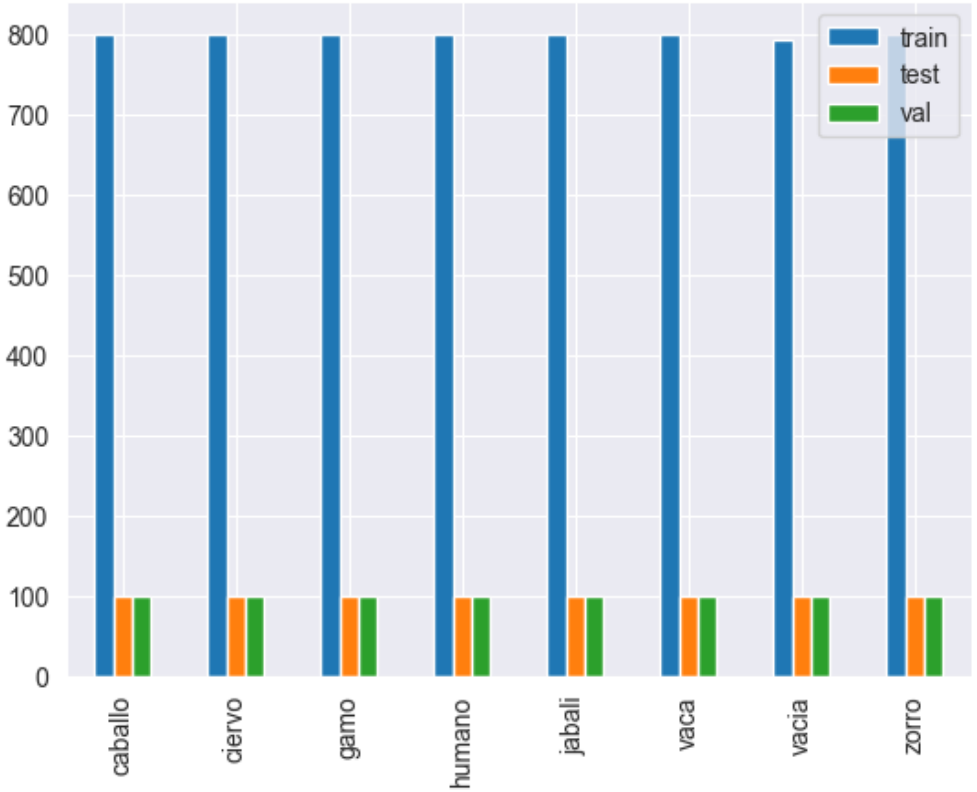

Figura 35: Distribución de clases

Fuente: Elaboración propia

7. Metodología

A continuación se va a presentar la metodología relativa a la generación de los diferentes modelos. Se abordarán las arquitecturas de CNN utilizadas en este proyecto, los hiperparámetros utilizados, las técnicas de *data augmentation*, el proceso de entrenamiento de los modelos, la optimización de parámetros y, finalmente, se presentarán las conclusiones y la selección de los modelos.

7.1. Arquitecturas

Primero se presentan las dos arquitecturas que se han aplicado al conjunto de datos: la red *MobileNetV2* y la *Arquitectura de Crohn*.

7.1.1. MobileNetV2

La red *MobileNetV2* consta de dos partes. La primera parte del modelo corresponde a las capas de *MobileNetV2* como base pre-entrenada. Esta base se carga utilizando la biblioteca Keras, y el código para cargarla se muestra en la Figura 11.2. La capa superior del modelo no es incluida, ya que el número de neuronas de esta capa depende del número de clases a clasificar en el modelo. Por lo tanto, se debe agregar por separado.

Para adaptar el modelo *MobileNetV2* a la tarea específica de clasificación de imágenes de fototrampeo de ocho clases de animales diferentes, se agregaron capas adicionales en la parte superior del modelo base pre-entrenado. Estas capas adicionales se encargan de realizar la clasificación final.

El entrenamiento de esta red se ha realizado en dos fases. En primer lugar se ha aplicado lo que llamamos un entrenamiento congelado (*frozen*), en el que todos los parámetros de la base se configuran como no entrenables. De esta forma, en este entrenamiento se conservarán los pesos precargados mediante *Transfer Learning*, que corresponden a los pesos de la competición *Imagenet*, y tan sólo se actualizarán los pesos de la capa superior añadida. En una segunda fase, se aplica un entrenamiento descongelado (*unfrozen*) con todos los parámetros del modelo entrenables.

En el código mostrado en la Figura 11.3, se pueden observar las capas agregadas después de la base. A continuación, se explican las diferentes capas agregadas y las razones por las que se han incluído:

- **GlobalAveragePooling2D**: Esta capa se utiliza para reducir la dimensionalidad de la salida de la base pre-entrenada. En lugar de trabajar con un tensor de múltiples dimensiones, se realiza un promedio global sobre cada canal del tensor, generando un vector unidimensional. Esto permite capturar las características más relevantes de la imagen de entrada.
- **Dropout**: Se han incluido capas de Dropout después de cada capa densa (Dense) para mitigar el sobreaprendizaje.
- **Dense**: Las capas densas son capas completamente conectadas, en las cuales cada neurona está conectada a todas las neuronas de la capa anterior. En este caso, se han agregado capas densas con diferentes tamaños, como 512, 256, 128, 64 y 32 neuronas respectivamente. La elección de estos tamaños se basa en la complejidad de la tarea de clasificación y en la capacidad del modelo para capturar características relevantes.
- **Activation**: Se ha utilizado la función de activación *relu* en las capas densas. La función de activación *relu* (Rectified Linear Unit) es una función no lineal que introduce no linealidades en la red y ayuda a aprender representaciones más complejas.

- **Dense (predictions)**: Esta capa final tiene un número de neuronas igual al número de clases en la tarea de clasificación. Utiliza la función de activación *softmax* para asignar probabilidades a cada clase. La clase con la probabilidad más alta será la predicción final del modelo.

7.1.2. *Arquitectura de Crohn*

La *Arquitectura de Crohn* fue explicada en el apartado 4.4.2, donde se explicó que está formado por bloques, y de las capas de las que se componen estos bloques. Para facilitar su creación se han implementado funciones para poder insertar estos bloques de forma más cómoda. Las funciones de los bloques se recogen en el código de la Figura 11.4.

La creación del modelo se consigue con el código de la Figura 11.5 que hace uso de las funciones anteriormente definidas.

Para adaptar la *Arquitectura de Crohn* a la tarea específica de clasificación del conjunto de datos específico, La última capa es del tamaño del número de clases del conjunto de datos.

7.2. Hiperparámetros

La etapa de entrenamiento de una red neuronal implica la configuración cuidadosa de diversos hiperparámetros que influyen en su proceso de aprendizaje y capacidad de generalización. Los hiperparámetros desempeñan un papel esencial en la optimización del rendimiento del modelo y su capacidad para resolver tareas específicas. A continuación, se describen los hiperparámetros utilizados en los modelos *MobileNetV2* y en la *Arquitectura de Crohn*, así como las razones detrás de sus elecciones.

7.2.1. MobileNetV2

El modelo MobileNetV2 se entrenó utilizando una combinación de técnicas de transferencia de aprendizaje y *fine tuning*. En la Figura 11.6 se muestran los valores de los hiperparámetros empleados. A continuación se explica cada uno de ellos:

- **Tamaño de Lote (BATCH_SIZE)**: Un lote de tamaño 32 se utilizó para el entrenamiento. Esto permite un equilibrio entre la eficiencia de cómputo y la estabilidad del entrenamiento.
- **Dimensiones de la Imagen (TARGET_IMG_WIDTH, TARGET_IMG_HEIGHT, TARGET_IMG_CHANNELS)**: Las imágenes de entrada se redimensionaron a 224x224 píxeles con 3 canales de color (RGB) para adaptarse a la entrada esperada de la red pre-entrenada MobileNetV2.
- **Épocas de Entrenamiento (NUM_EPOCHS)**: Se realizaron 50 épocas de entrenamiento para permitir que el modelo aprenda patrones complejos en los datos.
- **Monitorización de Entrenamiento (CHECKPOINT_MONITOR)**: El monitor de punto de control se estableció en 'val_accuracy' para guardar los pesos del modelo con la mejor precisión en el conjunto de validación.
- **Detención Temprana (EARLYSTOP_MONITOR, EARLYSTOP_PATIENCE)**: Se aplicó detención temprana utilizando 'val_loss' como monitor, con una paciencia de 15 épocas para evitar el sobreajuste.
- **Tasa de Aprendizaje (Learning Rate)**: La tasa de aprendizaje se ajustó automáticamente durante el entrenamiento utilizando un factor de reducción de 0.5 cuando no se observaron mejoras significativas ('val_loss') durante 5 épocas.
- **Número de Clases (NUM_CLASSES)**: Dado que el problema involucraba la clasificación de imágenes en 8 clases, el modelo final tenía una capa de salida con 8 neuronas.
- **Regularización (Dropout)**: Capas de dropout fueron incorporadas después de cada capa densa para mitigar el sobreajuste y fomentar una mejor generalización. El *Dropout* inicial utilizado ha sido de 0.2.

7.2.2. *Arquitectura de Crohn*

La *Arquitectura de Crohn* fue diseñado específicamente para la clasificación de imágenes en la tarea de detección de enfermedad de Crohn. En la Figura 11.7 se muestran los valores de los hiperparámetros empleados. A continuación se explica cada uno de ellos:

- **Tamaño de Lote (BATCH_SIZE)**: Similar al modelo MobileNetV2, se utilizó un tamaño de lote de 32 para el entrenamiento.

- **Dimensiones de la Imagen (TARGET_IMG_WIDTH, TARGET_IMG_HEIGHT, TARGET_IMG_CHANNELS)**: Las imágenes de entrada se redimensionaron a 224x224 píxeles con 3 canales de color (RGB) para ser coherentes con la arquitectura del modelo.
- **Épocas de Entrenamiento (NUM_EPOCHS)**: Se realizaron 100 épocas de entrenamiento para permitir una mayor exploración de patrones en los datos al no ser una red pre-entrenada y tener que calcular los pesos de todas las neuronas sin tener unos valores previos.
- **Monitorización de Entrenamiento (CHECKPOINT_MONITOR)**: Al igual que en MobileNetV2, se monitorizó 'val_accuracy' para guardar los pesos del modelo con la mejor precisión en el conjunto de validación.
- **Detención Temprana (EARLYSTOP_MONITOR, EARLYSTOP_PATIENCE)**: La detención temprana se aplicó con una paciencia de 25 épocas basada en el seguimiento de 'val_loss'.
- **Tasa de Aprendizaje (Learning Rate)**: Similar al modelo MobileNetV2, la tasa de aprendizaje se ajustó dinámicamente para mejorar la convergencia y evitar el sobreajuste.
- **Número de Clases (NUM_CLASSES)**: La *Arquitectura de Crohn* también tuvo una capa de salida con 8 neuronas debido a la naturaleza de la tarea.
- **Regularización (Dropout)**: Capas de dropout se incorporaron para prevenir el sobreajuste y mejorar la generalización del modelo. Sin embargo, para un primer experimento la tasa de *Dropout* utilizada es de 0 ya que la *Arquitectura de Crohn* original no hace uso de capas de *Dropout*.

La tasa de aprendizaje (*Learning Rate*) es un hiperparámetro crítico que afecta la velocidad y la calidad de la convergencia durante el entrenamiento. Para este proyecto, se realizó una búsqueda de tasa de aprendizaje y se encontró que un valor de 0.001 proporciona un equilibrio óptimo entre convergencia rápida y estabilidad.

El tamaño del lote (*batch size*) se estableció en 32. El entrenamiento se realizó en lotes para aprovechar la eficiencia computacional de las GPUs modernas y acelerar el proceso de cálculo de gradientes y actualización de pesos.

El resto pertenece a la configuración de los *callbacks* que permiten controlar el modelo guardado y la selección del modelo final.

En resumen, la elección de los hiperparámetros se realizó cuidadosamente para garantizar la convergencia eficiente y la capacidad de generalización de los modelos *MobileNetV2* y de la *Arquitectura de Crohn* en sus respectivas tareas de clasificación de imágenes. Estos valores se derivaron de experimentación y consideraciones específicas de cada modelo y problema.

7.3. *Data Augmentation*

Para mejorar el rendimiento del modelo y aumentar la capacidad de generalización, se aplicó la técnica de *Data Augmentation*. El *Data Augmentation* sólo se realiza en la fase de entrenamiento del modelo.

En el código mostrado en la Figura 62, se encuentra la secuencia de transformaciones aplicadas como parte del Data Augmentation. A continuación, se explican cada una de las transformaciones realizadas y cómo funcionan:

- **Resizing**: Esta transformación redimensiona las imágenes al tamaño objetivo. En este caso, las imágenes se redimensionan a una altura y ancho específico que son los de entrada del modelo base *MobileNetV2*, 224x224.
- **Rescaling**: Se aplica una escala a los valores de los píxeles de las imágenes para que estén en el rango [0, 1]. En el código, se divide cada valor de píxel por 255 para lograr esta normalización.
- **RandomFlip**: Esta transformación realiza un volteo horizontal aleatorio de las imágenes. En el código, se aplica un volteo horizontal con una probabilidad del 50 % para generar variedad en las orientaciones de las imágenes durante el entrenamiento.
- **RandomRotation**: Se aplica una rotación aleatoria a las imágenes en un rango determinado. En el código, se establece un rango de rotación de ± 0.1 radianes, lo que introduce variabilidad en la orientación de las imágenes y ayuda al modelo a ser más robusto frente a diferentes ángulos de captura.
- **RandomContrast**: Se aplica un factor de contraste aleatorio a las imágenes. En el código, se utiliza un factor de contraste de 0.5. Esta transformación aumenta o disminuye el contraste de las imágenes, lo que puede ayudar al modelo a capturar mejor las características relevantes.
- **RandomBrightness**: Se realiza un ajuste aleatorio del brillo de las imágenes. En el código, se utiliza un factor de brillo de 0.3 y un rango de valores entre 0 y 1. Esto proporciona variabilidad en las condiciones de iluminación durante el entrenamiento.
- **GaussianNoise**: Se agrega ruido gaussiano a las imágenes. En el código, se utiliza una desviación estándar de 0.1 para generar un nivel de ruido adecuado. Esta transformación ayuda al modelo a ser más robusto a pequeñas variaciones en los valores de píxeles.
- **RandomTranslation**: Esta transformación realiza una traslación aleatoria de las imágenes en sentido vertical y horizontal. En el código, se utiliza un factor de traslación de 1 en ambas dimensiones. Se emplea el modo de relleno *wrap* y la interpolación bilineal para mantener la consistencia en los bordes de la imagen al realizar la traslación.

En la Figura 36, se muestran muestras de imágenes generadas mediante el *Data Augmentation* correspondientes al conjunto de datos. El *Data Augmentation* más reconocible es el *Random Translation*.

Se optó por aplicar la técnica de *random translation* debido a la limitada cantidad de imágenes disponibles. Esta técnica simula la alteración de las imágenes de manera que parezca como si se hubieran recortado y pegado de forma un tanto abrupta. Aunque a primera vista esto podría no resultar visualmente atractivo para el observador humano, tiene un propósito claro en el contexto del aprendizaje automático.

Figura 36: Muestra del conjunto de datos de fototrampeo con *Data Augmentation*

Fuente: Elaboración propia

El uso de *random translation* beneficia al modelo en términos de generalización. Al cambiar la posición del objeto de interés en las imágenes, en este caso, la especie animal a clasificar, y al variar el fondo de la imagen de manera controlada, se introduce una mayor diversidad en el conjunto de datos. Esto ayuda al modelo a aprender características más robustas y a no depender en exceso de detalles específicos de ubicación o entorno. En otras palabras, el *random translation* amplía la capacidad del modelo para reconocer la especie en diferentes contextos y posiciones, lo que puede llevar a una mejora significativa en su capacidad de clasificación.

7.4. Entrenamiento del Modelo

Una vez que se ha completado la preparación de los datos, incluyendo la aplicación de la técnica de Data Augmentation, el siguiente paso crucial es el entrenamiento del modelo de clasificación de especies utilizando las imágenes de fototrampeo. En esta sección, describiremos el proceso de entrenamiento para cada una de las arquitecturas propuestas.

7.4.1. *MobileNetV2*

Como se ha comentado anteriormente el entrenamiento de la red basada en *MobileNetV2* se ha llevado a cabo en dos fases. En una primera fase, se lleva a cabo un entrenamiento *congelado*, donde la arquitectura se beneficia del concepto de *Transfer Learning*. De esta forma, durante esta fase del entrenamiento se mantienen todos los pesos precargados de la red, obtenidos mediante su entrenamiento en el conjunto de datos de *ImageNet* y únicamente se actualizan los pesos de las capas finales agregadas al modelo para adaptarlo al problema de clasificación multiclase que nos ocupa. En una segunda fase de entrenamiento, se realizó una fase de ajuste fino (*fine-tuning*), donde se entrenaron todas las capas del modelo, incluyendo aquellas con pesos preentrenados, adaptando así el modelo de manera más específica a nuestro conjunto de datos.

Ambos entrenamientos se gestionaron mediante *callbacks* que monitorearon el rendimiento y finalizaron el proceso si no se observó una mejora en la métrica específica durante un número definido de épocas. Cada época generó un modelo, pero el modelo final utilizado para las evaluaciones se eligió mediante el callback de *early checkpoint*, que almacenó el mejor modelo en función de la métrica de precisión (*accuracy*) en el conjunto de datos de validación.

El modelo guardado al finalizar el primer entrenamiento se cargó y se sometió al segundo entrenamiento en la fase de ajuste fino. El modelo final utilizado fue el mejor modelo obtenido después de este último entrenamiento.

7.4.2. *Arquitectura de Crohn*

Para la *Arquitectura de Crohn*, se realizó un solo entrenamiento, ya que no fue posible aplicar Transfer Learning debido a que esta red se desarrolló específicamente para un problema concreto y no se disponía de sus pesos preentrenados.

El proceso de finalización del entrenamiento y el almacenamiento del mejor modelo se gestionaron de la misma manera que en la arquitectura *MobileNetV2*, utilizando los mismos callbacks.

Sin embargo, en este caso, al tratarse de un único entrenamiento, el modelo final utilizado para las evaluaciones fue el mejor modelo obtenido al concluir este primer y único entrenamiento.

7.5. Optimización de Parámetros y Conclusiones

Para ambas arquitecturas de redes, se realizó una experimentación en la búsqueda de la mejor configuración para el conjunto de datos propuesto. Se realizaron variaciones en los hiperparámetros mencionados anteriormente y se monitoreó la convergencia de las métricas que se detallarán en el próximo apartado. Sin embargo, cabe destacar que la modificación de estos parámetros no condujo a cambios significativos en la eficiencia de la red. Lo que realmente demostró tener un impacto sustancial fue la adaptación del factor de *dropout* y la realización de un *data augmentation* mucho más exhaustivo.

Este hallazgo apunta a una primera conclusión importante: cuando los resultados no alcanzan los niveles deseados, uno de los factores críticos a considerar es la naturaleza del conjunto de datos en sí. La inclusión de técnicas de *data augmentation*, que generan imágenes más diversas y variadas, y la optimización del factor de *dropout* para promover la independencia entre las neuronas, ha demostrado ser eficaz en casos en los que el conjunto de datos es insuficiente o de baja calidad. Este enfoque se alinea con la literatura existente que destaca la importancia del *data augmentation* para mejorar el rendimiento de los modelos de visión por computador [43].

7.6. Selección de los modelos

Durante el proceso de entrenamiento de ambas redes, se han empleado los *callbacks* previamente mencionados para gestionar el proceso de entrenamiento de manera eficiente y garantizar que el modelo resultante sea el más adecuado.

Uno de los *callbacks* fundamentales es el *Checkpoint Callback*, el cual cumple la función de guardar el modelo en un punto determinado si este supera al mejor modelo guardado previamente en términos de métricas de validación, en este caso, el *Validation Accuracy*. Esto asegura que siempre contemos con el mejor modelo alcanzado hasta ese momento.

Por otro lado, el *Early Stopping Callback* es crucial para evitar el sobreajuste (*overfitting*) del modelo. Este *callback* monitorea el *Validation Loss* y detiene el entrenamiento si no se observa una mejora significativa después de un número específico de épocas. Esto es esencial para evitar que el modelo continúe entrenando cuando ha alcanzado su capacidad máxima de aprendizaje y, en su lugar, se esté adaptando demasiado a los datos de entrenamiento, lo que podría resultar en una disminución del rendimiento en datos no vistos.

En resumen, el proceso de selección del mejor modelo se lleva a cabo automáticamente durante el entrenamiento. El *Checkpoint Callback* garantiza que siempre tengamos el mejor modelo hasta ese punto, en función del *Validation Accuracy*. Además, el *Early Stopping Callback* evita que el entrenamiento continúe más allá de lo necesario, asegurando que el modelo final sea una representación óptima de las capacidades de la red neuronal. El resultado final es un modelo bien ajustado que puede evaluarse de manera efectiva en datos de validación y *test*.

8. Evaluación

En esta sección, se examinarán los resultados obtenidos por los modelos previamente seleccionados, con un enfoque especial en la evaluación de su rendimiento en un conjunto de datos completamente desconocido para ellos: el conjunto de *test*.

8.1. Resultados en test

El análisis de los resultados se realizará principalmente en función del rendimiento en el conjunto de *test*. Es importante destacar que el conjunto de entrenamiento ya ha sido utilizado por el modelo durante el proceso de entrenamiento para ajustar sus pesos, y el conjunto de validación se ha empleado para optimizar y refinar su rendimiento. El conjunto de *test*, en contraste, es completamente desconocido para el modelo y no ha sido utilizado previamente en ninguna etapa del entrenamiento.

A pesar de centrarnos principalmente en la evaluación del conjunto de *test*, también se presentarán datos sobre el rendimiento en los conjuntos de entrenamiento y validación. Esto nos permitirá obtener una visión completa de la convergencia y eficacia del proceso de entrenamiento.

Es importante tener en cuenta que, en el contexto de la evaluación, nos interesan varias métricas, como la exactitud (*accuracy*), la pérdida (*loss*), y posiblemente otras métricas específicas del problema, dependiendo de los objetivos y requisitos de la clasificación de especies en fototrampeo. Estas métricas proporcionarán información valiosa sobre el rendimiento del modelo en diferentes aspectos, como la capacidad de generalización y la capacidad para evitar el sobreajuste a los datos de entrenamiento.

En resumen, la evaluación se basa principalmente en el conjunto de *test*, que es independiente y nunca ha sido visto por el modelo. Esto garantiza una evaluación imparcial del rendimiento del modelo en condiciones del mundo real. Además, se proporcionarán datos de entrenamiento y validación para obtener una visión completa del proceso de entrenamiento y cómo ha influido en el rendimiento del modelo.

8.1.1. MobileNetV2 ($DROPOUT = 0.2$)

La evolución del primer entrenamiento (entrenamiento *frozen*) que se muestra en la Figura 37, muestra el progreso de la red *MobileNetV2* durante las diferentes épocas de entrenamiento. Se puede apreciar que el *accuracy* sobre los datos de entrenamiento aumenta gradualmente, alcanzando un valor de 0.8683 en la última época. El loss, por otro lado, disminuye a medida que avanza el entrenamiento, llegando a un valor de 0.3821 en la última época. Sin embargo, sobre los datos de validación no se obtienen los resultados esperados de un modelo que converge adecuadamente, el accuracy logrado en la última época es de 0.5562, y en la primera época era de 0.45500, no ha tenido una buena mejora, mientras que el *accuracy* de entrenamiento sí. La falta de mejora en la precisión en el conjunto de validación puede deberse a que las últimas capas del modelo no se ajustan lo suficiente para adaptarse a las características específicas de los datos de fototrampeo. Esto puede indicar que se requiere una adaptación más fina de las capas pre-entrenadas para mejorar el rendimiento en la tarea de clasificación de animales. Aún así, la falta de convergencia podría deberse a otros factores. El modelo guardado es el que corresponde a un mejor accuracy sobre el conjunto de datos de validación, en el que se consiguió un 0.57250 de accuracy.

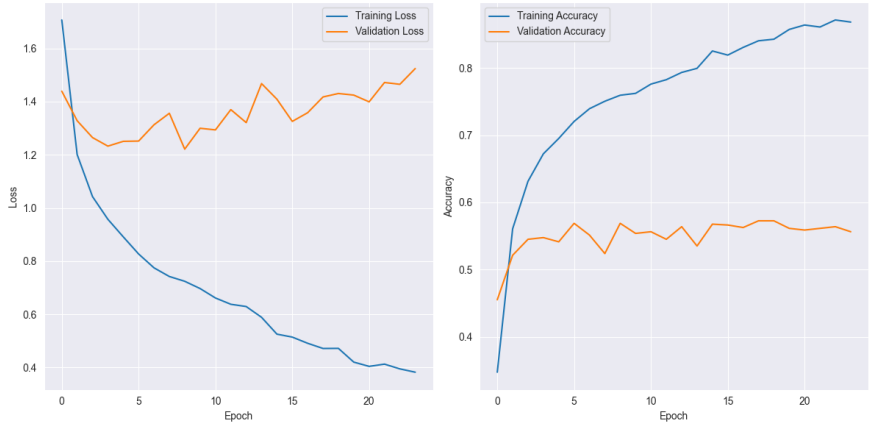

Figura 37: Evolución del entrenamiento *frozen* en *MobileNetV2 DROPOUT_FACTOR=0.2*

Fuente: Elaboración propia

Durante el segundo entrenamiento (entrenamiento *unfrozen*, cuya evolución puede verse en la Figura 38, se observó que sobre el conjunto de datos de entrenamiento hubo una mejora muy pequeña, mientras que sobre el conjunto de validación no hubo ningún cambio significativo, lo que indica que el modelo ya había llegado al sobreajuste en el entrenamiento anterior. El sobreajuste esuna preocupación común en el *deep learning*, se refiere a la situación en la que el modelo se ajusta demasiado a los datos de entrenamiento, capturando incluso el ruido en lugar de los patrones realmente importantes. En este caso, el hecho de que el rendimiento sobre el conjunto de entrenamiento mejore mientras que no se observe un progreso correspondiente en el conjunto de validación sugiere que el modelo ha agotado su capacidad de generalización y ha llegado al sobreaprendizaje.

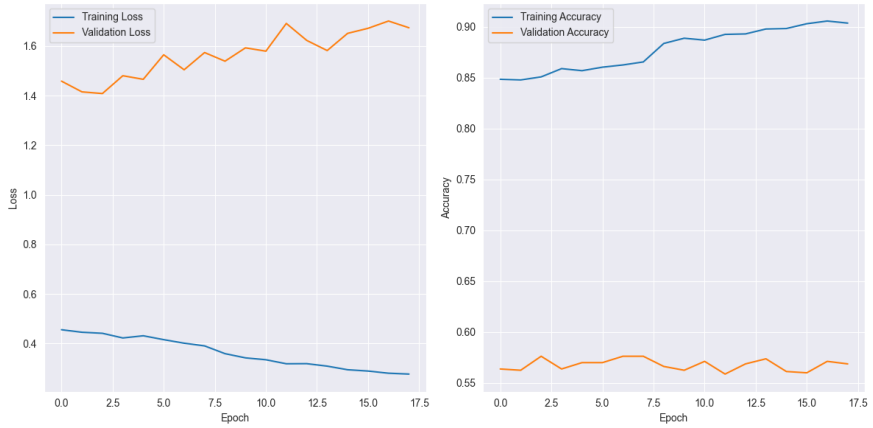

Figura 38: Evolución del entrenamiento *unfrozen* en *MobileNetV2 DROPOUT_FACTOR=0.2*

Fuente: Elaboración propia

En este caso, el *accuracy* sobre los datos de test fue de 68.69 %, mejorando apenas unas décimas en comparación con el *accuracy* obtenido en el entrenamiento *frozen*, lo que subraya la necesidad de abordar el sobreajuste para lograr un mejor rendimiento. La precisión obtenida indica que el modelo clasificó correctamente aproximadamente el 68.69 % de las imágenes en el conjunto de *test*, mejorando solo unas décimas en comparación con el entrenamiento *frozen*. Aunque esta precisión es mejor que el azar (que sería del 50 % en un problema de clasificación binaria), todavía hay margen

para mejorar el rendimiento del modelo. Por otro lado el *Loss* obtenido es de 1.0208.

La Figura 39 muestra la matriz de confusión generada a partir del conjunto de datos de *test* después de completar el entrenamiento *unfrozen* de la red basada en *MobileNetV2*. Esta matriz de confusión resalta los verdaderos positivos a lo largo de su diagonal principal. A primera vista, parece que el modelo está realizando una clasificación adecuada, ya que esta diagonal principal se ve en un mayor contraste, es decir, más poblada. Sin embargo, basarse únicamente en la matriz de confusión y el porcentaje de precisión (*accuracy*) puede ser engañoso.

Es importante tener en cuenta que, a pesar de la aparente buena disposición de la matriz de confusión, los resultados generales pueden no ser tan sólidos. El valor de *accuracy* obtenido no es muy alto, y las gráficas que muestran la evolución del entrenamiento muestran que el modelo ha sufrido de *overfitting*. Este fenómeno ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a datos no vistos anteriormente.

Para comprender mejor la situación, es necesario considerar la posibilidad de que el conjunto de *validación* sea muy similar al conjunto de entrenamiento, lo que podría llevar a una evaluación optimista del modelo. En estudios de ecología que involucran imágenes de fototrampeo, este escenario es bastante común. Las ráfagas de fotografías capturadas en secuencia pueden resultar en imágenes similares presentes en diferentes subconjuntos, lo que significa que el modelo podría haber visto estas imágenes previamente. Esto contradice el propósito de un conjunto de validación y *test*, que debería contener datos completamente nuevos para evaluar la capacidad de generalización del modelo.

En resumen, aunque la matriz de confusión sugiere un buen rendimiento, es esencial considerar cuidadosamente los posibles efectos del sobreaprendizaje y la similitud entre los conjuntos de entrenamiento y *test* al interpretar estos resultados. Estos desafíos subrayan la importancia de abordar de manera crítica la calidad y diversidad del conjunto de datos en estudios de ecología basados en imágenes de fototrampeo.

En la Figura 40, se presentan algunas predicciones realizadas por la red *MobileNetV2* sobre el conjunto de datos de *test*. Estas predicciones muestran la capacidad de la red para clasificar correctamente diferentes clases de animales. Las imágenes mostradas han sido seleccionadas de forma aleatoria, y hay tanto imágenes clasificadas correctamente como de forma errónea. La confianza con la que clasifica tanto las imágenes correctas como erróneas parece no ser significativa ni indicativa, pues hay imágenes que acierta con una confianza alta y otras con una baja. Del mismo modo pasa con las imágenes que no ha clasificado bien.

Con esto se puede concluir que, como las gráficas de evolución del entrenamiento mostraban, el modelo ha sobreaprendido, los factores por los que puede haber llegado el modelo al sobreaprendizaje son varios.

Uno de estos factores podría ser el tamaño limitado del conjunto de datos, no es un conjunto de datos muy grande teniendo 1000 imágenes de cada clase para el entrenamiento. Aumentar la cantidad de datos de entrenamiento podría permitir al modelo aprender patrones más robustos y generalizables.

Por otro lado, otro factor podría ser la dificultad de las imágenes. Estas imágenes no se han sometido a un proceso de detección previo en el que se restringe la imagen a clasificar a la región de interés, en este caso del animal, lo que ayudaría a disminuír el error. De este modo, sin aplicar una

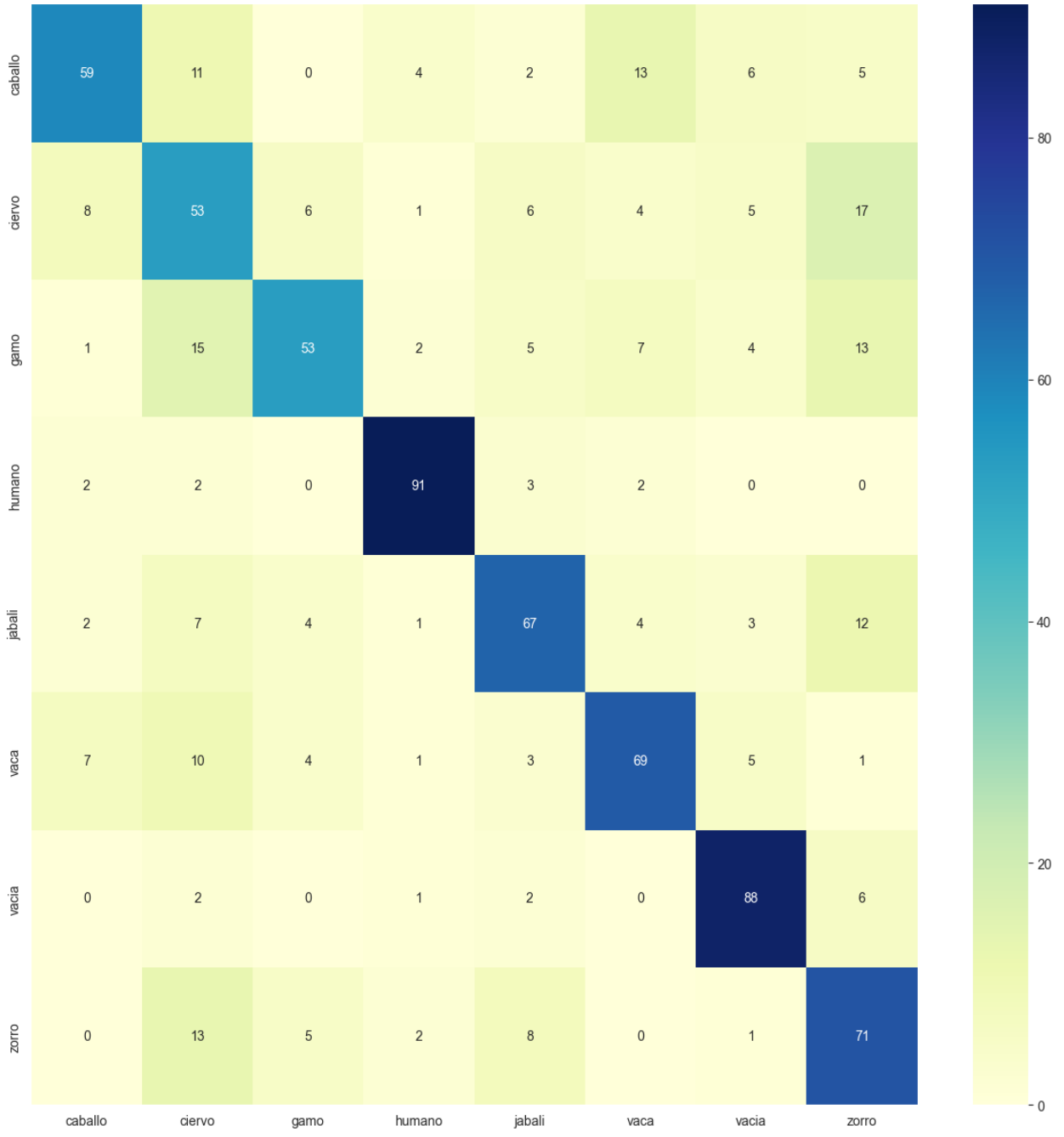

Figura 39: Matriz de confusión tras el entrenamiento *unfrozen* en *MobileNetV2 DROPOUT_FACTOR=0.2* sobre el conjunto de datos de test

Fuente: Elaboración propia

detección previa lo que se analiza es la imagen completa que incluye tanto el fondo de la imagen como el animal. A veces el animal a clasificar puede estar escondido en la imagen de forma que sea casi invisible. Detectar el animal en las imágenes antes de pasárselas al modelo de clasificación podría ayudar al modelo a reconocer y clasificar mejor al animal.

Otro de estos factores también podría ser las características del conjunto de datos en el que

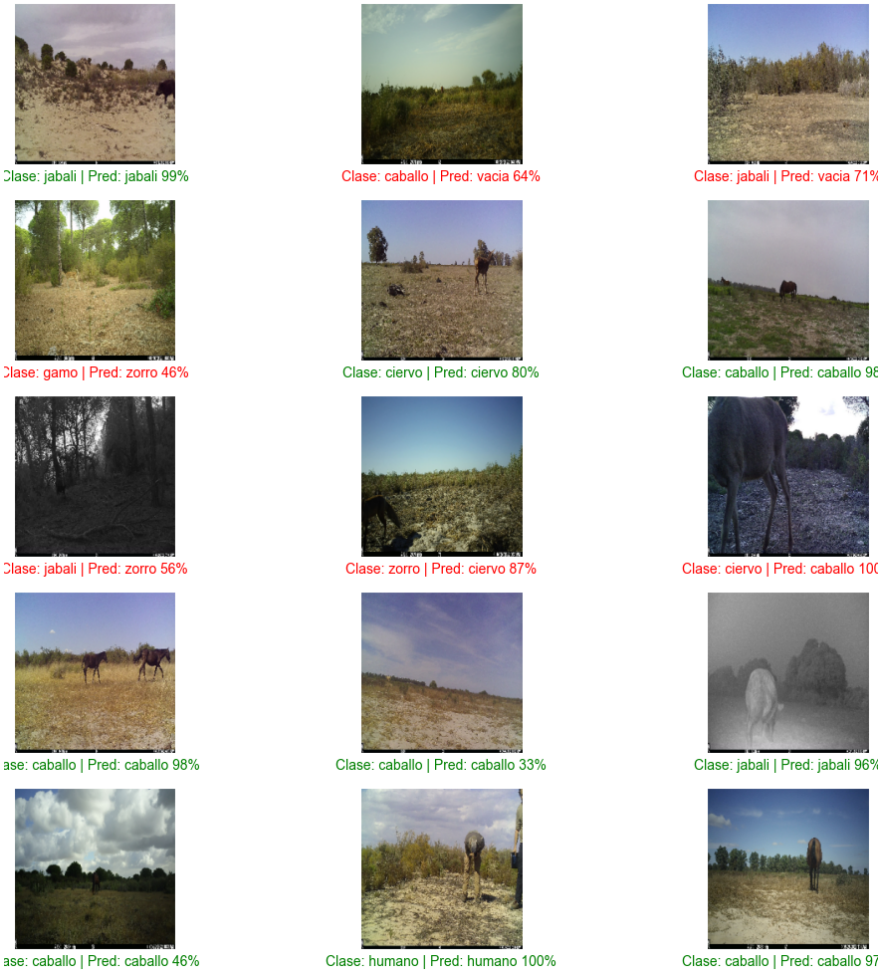

Figura 40: Predicciones con *MobileNetV2*

Fuente: Elaboración propia

las imágenes no son independientes entre ellas. Para que los conjuntos de datos de entrenamiento, validación y *test* funcionen como se espera deben ser independientes y reflejar la realidad. Si existe una alta correlación entre estos conjuntos entonces se deja parte del espacio sin explorar. En este caso, al no tener suficiente información no se han tenido en cuenta las ráfagas de las fotografías para realizar la separación así como otros factores importantes para realizar un conjunto de datos tan diverso como se pueda y los subconjuntos de entrenamiento, validación y *test* independientes.

Otra posible mejora es ajustar los hiperparámetros del modelo y del proceso de entrenamiento. Por ejemplo, se podría considerar una tasa de aprendizaje más baja para un entrenamiento más estable y una mayor capacidad de exploración del espacio de soluciones.

Además, es importante destacar que hacer *Transfer Learning* ayuda a ahorrar el tiempo de entrenamiento de las primeras etapas, y con hacerle un pequeño entrenamiento ya se puede ajustar un poco el modelo a los datos del problema. Aunque con un conjunto de datos tan pequeño e imágenes tan complejas el modelo no consigue generalizar bien. Una posible mejora de los hiperparámetros sería aumentar el Dropout. El introducido en este experimento es de 0.2, con un mayor dropout se ayudaría a las neuronas a tener más independencia entre ellas y por lo tanto lograr una mejor

generalización.

En resumen, tanto el entrenamiento solo sobre las últimas capas como el entrenamiento sobre todas las capas del modelo presentaron desafíos en términos de sobreajuste. Esto indica que se requiere una exploración más profunda de técnicas de regularización y ajuste de hiperparámetros para mejorar el rendimiento del modelo y lograr una mejor generalización en la clasificación de animales en imágenes de fototrampeo.

8.1.2. MobileNetV2 ($DROPOUT = 0.5$)

En línea con las observaciones realizadas en el experimento anterior, se ha llevado a cabo un nuevo experimento destinado a abordar el sobreaprendizaje. En este caso, la variable de interés fue el factor de *dropout*. Es fundamental destacar que, para una comparación justa y significativa entre dos modelos con diferentes niveles de *dropout*, ningún otro parámetro se modificó. Esto garantiza que cualquier mejora o empeoramiento en el rendimiento del modelo pueda atribuirse directamente al ajuste del *dropout*, sin interferencias de otros factores.

El aumento del *dropout* se considera una estrategia para mitigar el sobreaprendizaje, ya que introduce más aleatoriedad y desconexiones temporales durante el entrenamiento, lo que podría ayudar al modelo a generalizar mejor. El objetivo principal de este cambio es mejorar las métricas de *accuracy* tanto en el conjunto de validación como en el conjunto de *test*. Además, se busca lograr una mejor convergencia en las gráficas que representan la evolución del entrenamiento.

Este experimento proporcionará información valiosa sobre la influencia del factor de *dropout* en el rendimiento de la red, lo que contribuirá a la comprensión más profunda de cómo evitar el sobreaprendizaje y mejorar la capacidad de generalización del modelo.

La Figura 41 presenta un análisis detallado de la evolución del entrenamiento de la red *MobileNetV2* durante las diversas épocas. Durante el proceso de entrenamiento, se observa un aumento gradual en la métrica de *accuracy* en los datos de entrenamiento, alcanzando un valor de 0.7051 en la última época. En contraste, la métrica de pérdida (*loss*) disminuye a medida que avanza el entrenamiento, llegando a un valor de 0.8813 en la última época. Estos indicadores sugieren que la red está aprendiendo de manera efectiva de los datos de entrenamiento, mejorando su capacidad para clasificar las especies.

Pero la situación cambia cuando se evalúa el modelo en el conjunto de datos de validación. El *accuracy* alcanzado en la última época es de 0.5375, con un valor inicial de 0.2450 en la primera época. Esta falta de mejora en la precisión en el conjunto de validación podría deberse a varias razones. Es posible que las últimas capas del modelo no se adapten lo suficiente para capturar las características específicas de los datos de fototrampeo. Esto podría indicar la necesidad de un ajuste más fino de las capas preentrenadas para mejorar el rendimiento en la tarea de clasificación de animales. Sin embargo, a diferencia de los resultados obtenidos en el experimento anterior, en esta ocasión la evolución de las gráficas muestra indicios de una mejor convergencia. A pesar de esta mejora, aún es evidente la presencia de sobreaprendizaje, lo que se refleja en un patrón en el que, después de un número determinado de épocas, el *accuracy* en el conjunto de validación parece estancarse, mientras que el del conjunto de entrenamiento continúa mejorando.

Este fenómeno sugiere que aunque el aumento del factor de *dropout* ha contribuido a reducir el

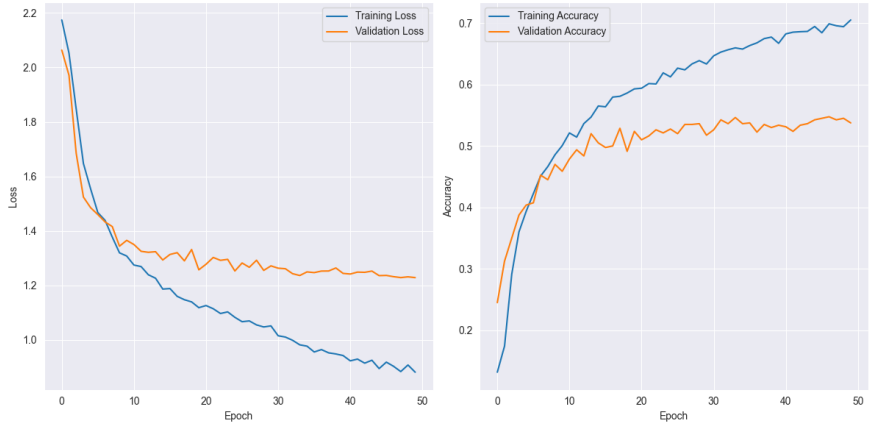

Figura 41: Evolución del entrenamiento *frozen* en *MobileNetV2 DROPOUT_FACTOR=0.5*

Fuente: Elaboración propia

sobreaprendizaje, aún persiste en cierta medida. Es importante destacar que abordar el sobreaprendizaje en su totalidad puede ser un desafío, ya que puede estar influenciado por varios factores, como la calidad y cantidad de datos, la complejidad del modelo y la elección de otros hiperparámetros. A pesar de ello, este experimento ha demostrado que el ajuste del factor de *dropout* puede ser una estrategia efectiva para mejorar la convergencia y la capacidad de generalización del modelo.

Durante el entrenamiento *unfrozen*, cuya evolución puede verse en la Figura 42, se observó que sobre el conjunto de datos de entrenamiento hubo una mejora muy pequeña, mientras que sobre el conjunto de validación no hubo ningún cambio significativo, lo que indica que el modelo ya había llegado al sobreajuste en el entrenamiento anterior. El accuracy sobre los datos de test fue de 66.83 %, mejorando apenas unas décimas en comparación con el accuracy obtenido en el entrenamiento *frozen*.

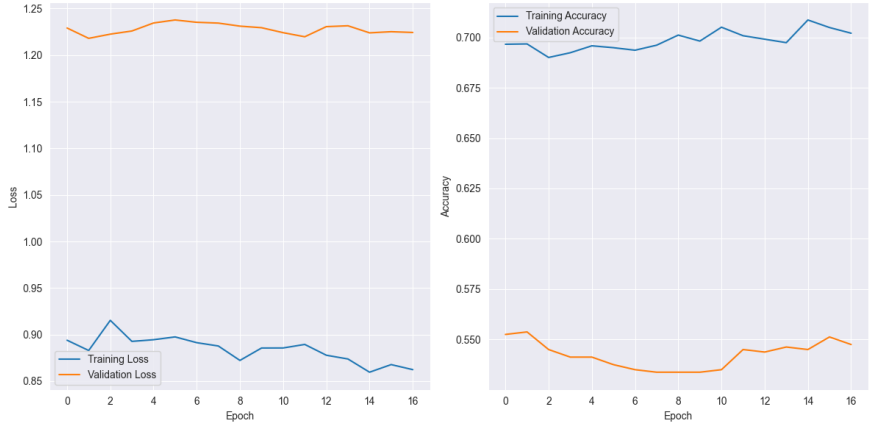

Figura 42: Evolución del entrenamiento *unfrozen* en *MobileNetV2 DROPOUT_FACTOR=0.5*

Fuente: Elaboración propia

El *accuracy* obtenido sobre el conjunto de *test* indica que el modelo clasificó correctamente aproximadamente el 66.71 % de las imágenes en el conjunto, empeorando solo unas décimas en comparación con el primer experimento, con un menor factor de *dropout*. A contiuación se mostrarán

otras visualizaciones para poder realizar una evaluación y análisis de la mejora o el empeoramiento del modelo con el nuevo ajuste del valor del factor de *dropout*.

La Figura 43 muestra la matriz de confusión generada a partir de la evaluación del modelo en el conjunto de datos de *test*. Como era de esperar, la matriz presenta una diagonal principal que representa los verdaderos positivos esperados para cada clase. Sin embargo, se observan algunas áreas de confusión destacadas en la matriz. En particular, la clase *vaca* se clasifica erróneamente como *caballo* en un número significativo de ocasiones. Además, se observa una alta tasa de confusión entre las clases *ciervo* y *gamo*. Esta última confusión podría considerarse más comprensible, dado que estas dos especies son bastante similares, incluso para el ojo humano. En contraste, el *ciervo* apenas lo confunde por un *gamo*. Las clases que mejor distingue son *humano* y *vacía* que son las que aparentemente se diferencian más de las otras clases y especies.

Para mejorar la precisión de la clasificación en estas clases que confunde con facilidad, se podría considerar la inclusión de un mayor número de imágenes de *ciervo* y *gamo* y de *vaca* y *caballo* en el conjunto de datos de entrenamiento. Esto ayudaría al modelo a aprender y distinguir mejor las características distintivas de estas especies, lo que podría conducir a una clasificación más precisa.

En la Figura 44, se presentan algunas predicciones realizadas por la red *MobileNetV2* sobre el conjunto de datos de *validación*. Las imágenes mostradas han sido seleccionadas de forma aleatoria, y hay tanto imágenes clasificadas correctamente como de forma errónea. Del mismo modo que pasaba en el experimento anterior la confianza con la que son clasificadas tanto las imágenes correctas como erróneas no es significativa, pues hay imágenes que acierta y falla con una confianza alta y otras con una baja.

En base a los resultados obtenidos, es evidente que el modelo ha experimentado sobreaprendizaje. El ajuste del factor de *dropout*, que inicialmente parecía estar contribuyendo a una mejor convergencia durante el entrenamiento, no ha generado una mejora significativa en la precisión. En realidad, el *accuracy* en el conjunto de *test* ha disminuido ligeramente, aunque sólo en unas décimas.

Esto sugiere que podría ser beneficioso realizar una experimentación más exhaustiva, explorando diferentes valores de diversos hiperparámetros. Sin embargo, es importante destacar que los resultados obtenidos también ponen de manifiesto una limitación fundamental: la falta de imágenes y datos de alta calidad. Este problema dificulta la creación de un conjunto de datos robusto y diverso para el entrenamiento del modelo.

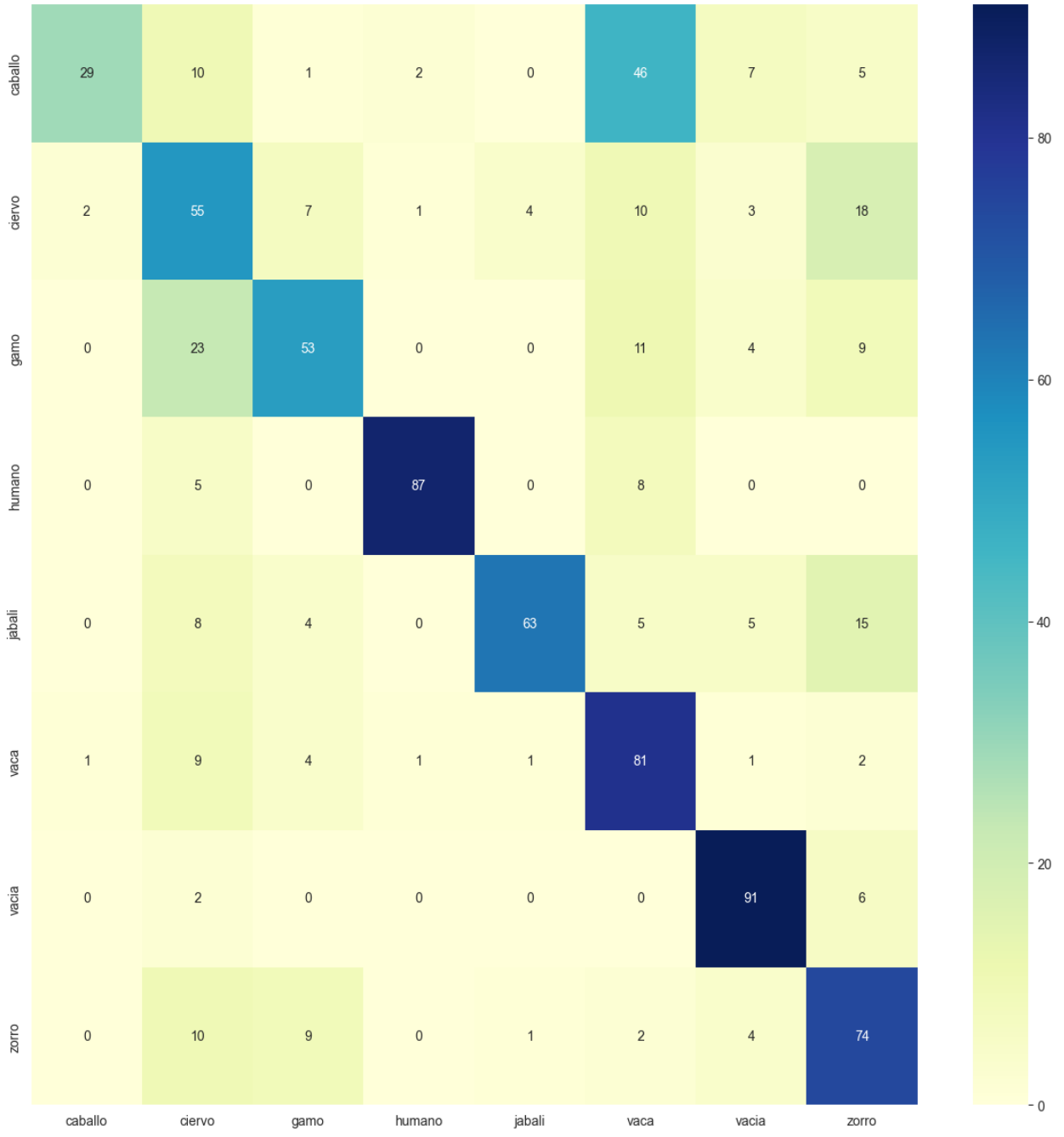

Figura 43: Matriz de confusión tras el entrenamiento *unfrozen* en *MobileNetV2 DROPOUT_FACTOR=0.5* sobre el conjunto de datos de *test*

Fuente: Elaboración propia

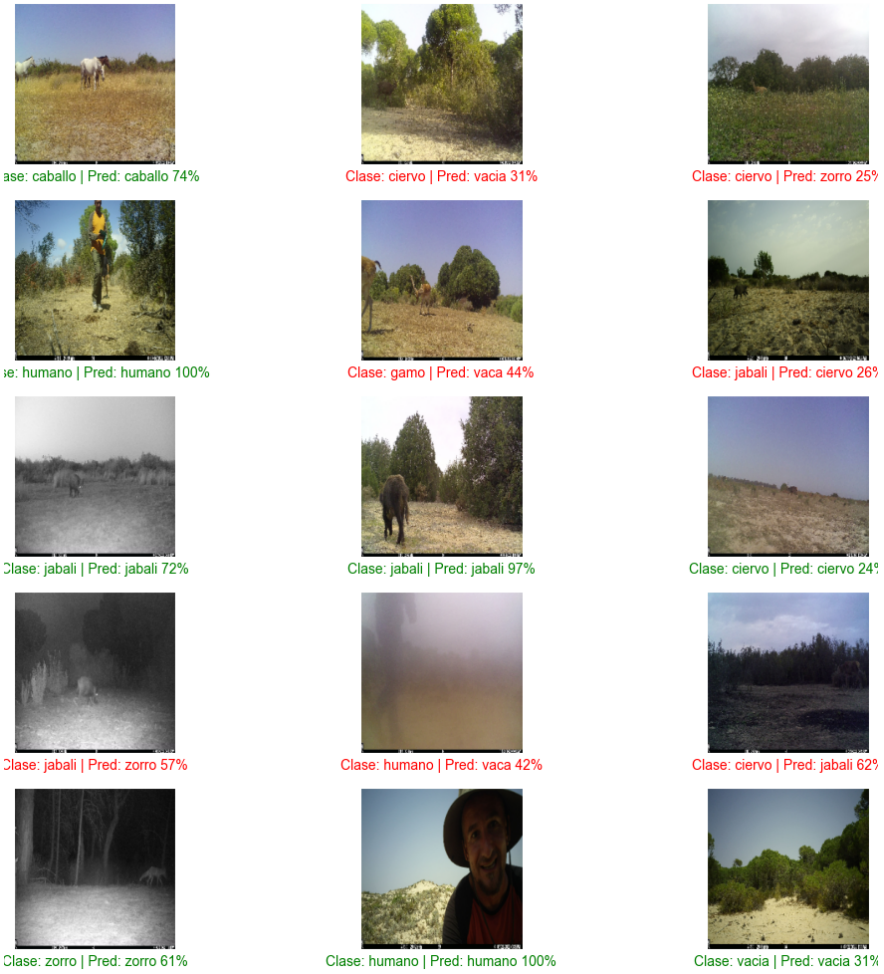

Figura 44: Predicciones con *MobileNetV2*

Fuente: Elaboración propia

8.1.3. *Arquitectura de Crohn* ($DROPOUT = 0$)

La arquitectura de la red *MobileNetV2*, junto con las capas superiores añadidas, contiene un total de 3,088,680 parámetros que se entrenan a lo largo de los dos entrenamientos. Este número de parámetros puede considerarse alto en relación con la cantidad limitada de imágenes disponibles, lo que podría estar contribuyendo al problema de sobreaprendizaje.

Para abordar este desafío y explorar una solución potencial al sobreaprendizaje, se ha optado por considerar una arquitectura alternativa que hemos decidido llamar *Arquitectura de Crohn* en la Sección 4.4.2. Esta arquitectura, en contraste, contiene un total de 599,168 parámetros, lo que representa una reducción significativa en comparación con la arquitectura inicial de *MobileNetV2*, llegando a ser aproximadamente el 20 % de la cantidad de parámetros originales.

Como se mencionó anteriormente al describir la *Arquitectura de Crohn*, ésta puede ser utilizada tanto con como sin *dropout*. Se iniciará un experimento utilizando un factor de *dropout* de 0, es decir, como si no se aplicase ningún *dropout* en el modelo. Este enfoque permitirá evaluar cómo se comporta esta arquitectura en comparación con la anterior y si la reducción significativa en el número de parámetros contribuye a una mejora en el rendimiento y la generalización del modelo.

La Figura 45 proporciona una visualización de la evolución del entrenamiento de la red basada en la *Arquitectura de Crohn* a lo largo de las múltiples épocas. Durante este proceso de entrenamiento, se observa un aumento gradual en la métrica de *accuracy* en los datos de entrenamiento, llegando a un valor de 0.8337 en la última época. Adicionalmente, la métrica de pérdida (*loss*) disminuye progresivamente a medida que avanza el entrenamiento, alcanzando un valor de 0.5042 en la última época. Estos indicadores sugieren que la red está aprendiendo de manera efectiva a partir de los datos de entrenamiento, mejorando su capacidad para clasificar las especies. Además, los resultados muestran una mejora en comparación con la arquitectura anterior, *MobileNetV2*.

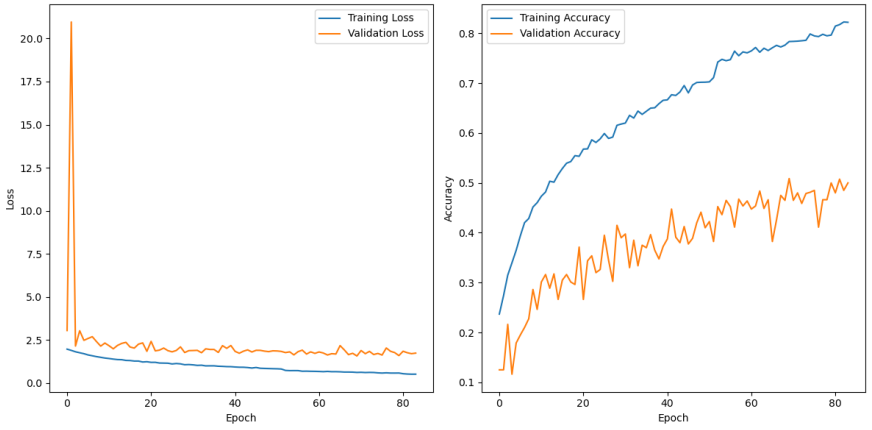

Figura 45: Evolución del entrenamiento en la *Arquitectura de Crohn DROPOUT_FACTOR=0*

Fuente: Elaboración propia

Sin embargo, la situación cambia cuando se evalúa el modelo en el conjunto de datos de validación. El *accuracy* obtenido en la última época es de 0.5000, con un valor inicial de 0.1250 en la primera época. A pesar de que ambas gráficas, tanto la del *accuracy* en los datos de entrenamiento como en los de validación, continúan mejorando, la brecha entre estas métricas se amplía con el tiempo. Esta disparidad indica un posible problema de sobreaprendizaje, ya que el modelo se

vuelve cada vez más competente en la tarea de entrenamiento, pero no logra generalizar de manera efectiva sobre los datos de validación al mismo ritmo.

El *accuracy* sobre los datos de *test* fue de 62.08 %, empeorando los valores de las métricas logrados con los experimentos anteriores.

Este entrenamiento debería haberse detenido automáticamente mediante el *callback* o al finalizar el número total de épocas establecido como hiperparámetro. Sin embargo, varios factores influyeron en la decisión de interrumpir manualmente el proceso de entrenamiento. Uno de los principales motivos es que, a diferencia de la arquitectura de *MobileNetV2*, que completó sus dos entrenamientos en aproximadamente 1 día (con una duración de alrededor de 2 minutos por época), la *Arquitectura de Crohn* demostró ser mucho más costosa en términos de tiempo. Cada época de entrenamiento requería alrededor de 18 minutos, lo que resultó en más de 2 días de entrenamiento para un solo experimento.

Durante el monitoreo del progreso del entrenamiento y al observar los síntomas de sobreaprendizaje, se tomó la decisión de detener manualmente el proceso. Esto se hizo con el propósito de realizar un nuevo experimento para abordar el problema de sobreaprendizaje y buscar soluciones alternativas. El entrenamiento se paró manualmente al alcanzar la época 85.

La Figura 46 muestra la matriz de confusión generada a partir de la evaluación del modelo en el conjunto de datos de *test*. Como era de esperar, la matriz presenta una diagonal principal que representa los verdaderos positivos esperados para cada clase.

No obstante, al examinar detenidamente la matriz de confusión, aún se pueden identificar áreas de confusión notables. Estas áreas y patrones de aciertos y errores coinciden en gran medida con los observados en el experimento anterior, a pesar de la arquitectura de red diferente utilizada. Las clases *vacía* y *humano* muestran resultados notables, siendo clasificadas correctamente en la mayoría de los casos, mientras que otras clases experimentan una mayor tasa de confusión.

En particular, se observan confusiones significativas entre la clase *vaca* y *caballo*, así como entre las clases *ciervo* y *gamo*. Para abordar este problema, una solución potencial podría ser la misma que se propuso en el experimento anterior: aumentar el número de imágenes disponibles para estas clases que tienden a ser confundidas con mayor frecuencia. Esto podría ayudar al modelo a aprender y distinguir mejor entre estas especies similares.

En la Figura 47, se presentan algunas predicciones realizadas por la *Arquitectura de Crohn* sobre el conjunto de datos de *test*. Las imágenes mostradas han sido seleccionadas de forma aleatoria, y hay tanto imágenes clasificadas correctamente como de forma errónea. Del mismo modo que pasaba en los experimentos anteriores la confianza con la que son clasificadas tanto las imágenes correctas como erróneas no es significativa, pues hay imágenes que acierta y falla con una confianza alta y otras con una baja.

En resumen, a partir de este experimento se puede concluir que, aunque la convergencia durante el entrenamiento parecía más prometedora en comparación con los experimentos anteriores, los resultados finales de precisión en el conjunto de datos de prueba no han experimentado una mejora significativa. Además, la matriz de confusión muestra similitudes con las confusiones observadas en experimentos previos.

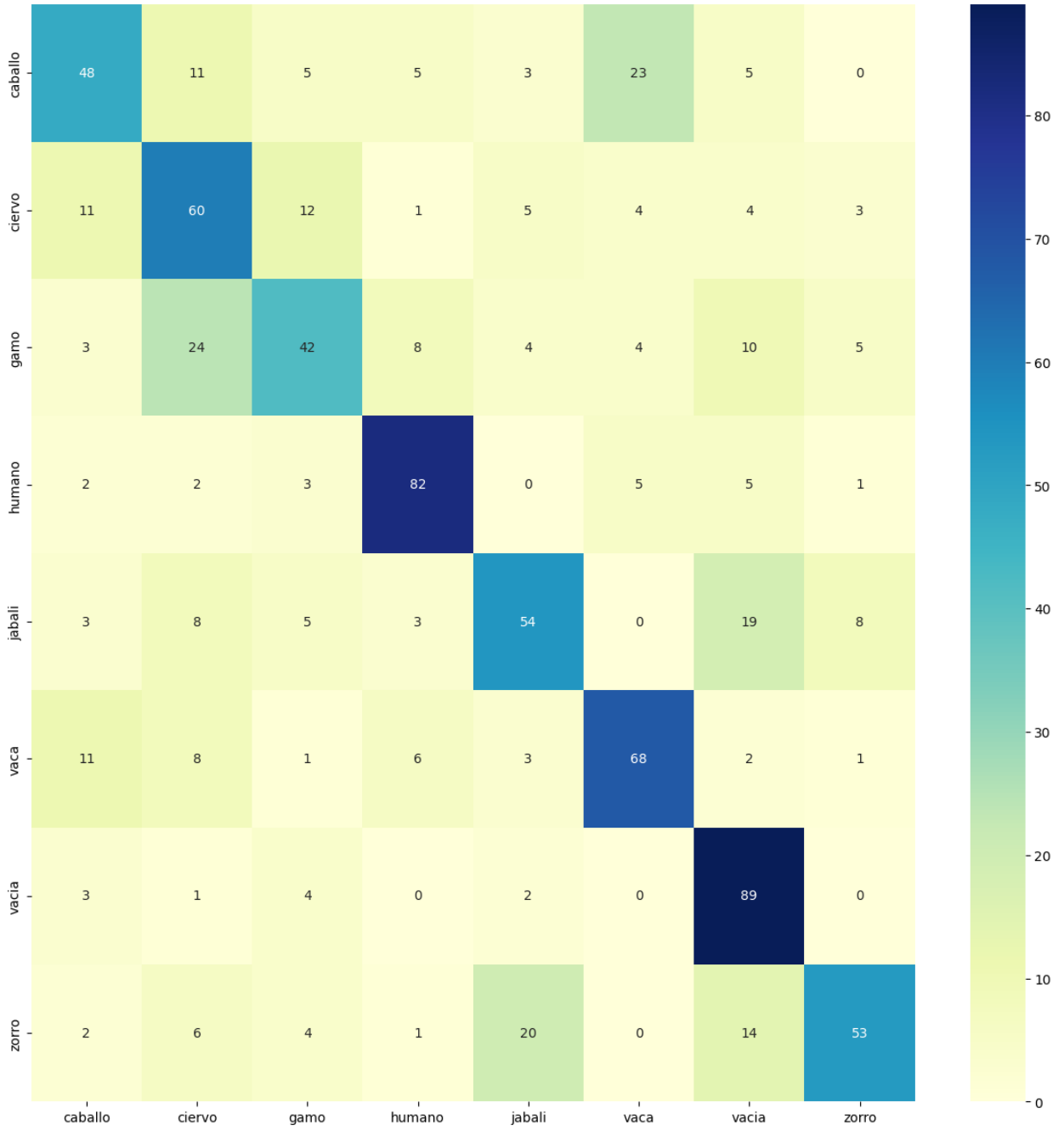

Figura 46: Matriz de confusión en la *Arquitectura de Crohn DROPOUT_FACTOR=0* sobre el conjunto de datos de test

Fuente: Elaboración propia

Este patrón resalta nuevamente la persistencia del sobreaprendizaje, que parece ser un desafío difícil de superar mediante simples ajustes de hiperparámetros o cambios en la arquitectura del modelo. Por lo tanto, es plausible considerar que la raíz del problema podría estar relacionada con la calidad y la distribución de los datos en el conjunto de datos, así como con la partición de los mismos en conjuntos de entrenamiento, validación y prueba.

Figura 47: Predicciones con la *Arquitectura de Crohn DROPOUT_FACTOR=0.2*

Fuente: Elaboración propia

Con el fin de confirmar que el problema radica en el sobreaprendizaje, se llevará a cabo un último experimento en el que se aumentará el valor del factor de *dropout*, que actualmente se encuentra en 0, en un intento por mitigar aún más el sobreaprendizaje.

8.1.4. *Arquitectura de Crohn* ($DROPOUT = 0.2$)

Para estudiar si se podría reducir el sobreaprendizaje al modificar un hiperparámetro en la red basada en la *Arquitectura de Crohn*, se estableció un factor de *dropout* de 0,2. A continuación, se presenta la evolución del entrenamiento y los resultados obtenidos.

La Figura 48 ilustra la progresión del entrenamiento de la *Arquitectura de Crohn* con un factor de *dropout* configurado en 0,2. Durante este proceso de entrenamiento, se observa un aumento gradual en la métrica de *accuracy* en los datos de entrenamiento. Sin embargo, no se observa ninguna mejora en la métrica de precisión en los datos de validación. Además, la métrica de *loss* disminuye de manera constante en los datos de entrenamiento, pero no presenta mejoras en los datos de validación.

Debido a la ausencia de mejoras en los datos de validación en términos de métricas de *loss* y *accuracy*, el entrenamiento se detuvo manualmente en la época 36. Esta decisión se tomó porque la *Arquitectura de Crohn* requiere un tiempo considerable para entrenarse, y después de 36 épocas no se observó ningún indicio de mejora. En resumen, el cambio en el factor de *dropout* no generó ninguna mejora en el rendimiento de la red, ya que no se evidencia ningún aprendizaje significativo.

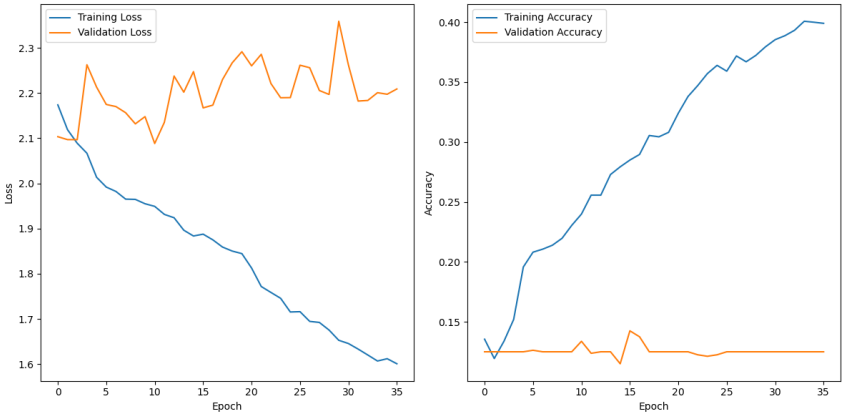

Figura 48: Evolución del entrenamiento en la *Arquitectura de Crohn DROPOUT_FACTOR=0.2*

Fuente: Elaboración propia

El *accuracy* en los datos de *test* fue de 19,77 %, lo que representa un deterioro con respecto a los resultados obtenidos en los experimentos anteriores y confirma la incapacidad de la red para realizar aprendizaje significativo.

La Figura 49 muestra la matriz de confusión generada a partir de la evaluación del modelo en el conjunto de datos de *test*. En este experimento, la matriz no muestra una diagonal principal que represente los verdaderos positivos esperados para cada clase. Más bien, la mayoría de las clases se clasifican erróneamente como *ciervo*. A pesar de la baja tasa de aciertos y el rendimiento deficiente en el entrenamiento, es difícil entender por qué ocurre esto. Estos resultados subrayan aún más las limitaciones de la red y destacan la necesidad de abordar los desafíos fundamentales en relación con el *dataset* y la separación de los datos de manera adecuada para futuros experimentos.

En la Figura 50, se presentan algunas de las predicciones realizadas por la *Arquitectura de Crohn* sobre el conjunto de datos de *test*. Estas imágenes se han seleccionado de forma aleatoria y muestran tanto ejemplos en los que la red clasifica correctamente como casos en los que comete errores. Sin embargo, la mayoría de las imágenes se han clasificado de manera incorrecta. Notablemente, tanto en las imágenes clasificadas correctamente como en las incorrectas, la puntuación de confianza en las predicciones es bastante similar, oscilando alrededor del 17 al 18 %. Este patrón sugiere que la red tiene dificultades para realizar predicciones precisas y tiende a asignar puntajes de confianza bajos en general.

Este experimento, que implicó ajustar el factor de *dropout* a 0,2 en la red basada en la *Arquitectura de Crohn*, arrojó resultados desalentadores. A pesar del cambio en el hiperparámetro, el rendimiento de la red no mejoró en absoluto.

A pesar de los esfuerzos por reducir el sobreaprendizaje mediante la modificación del factor de *dropout*, los resultados no mostraron mejoría. El modelo todavía luchaba por generalizar a partir

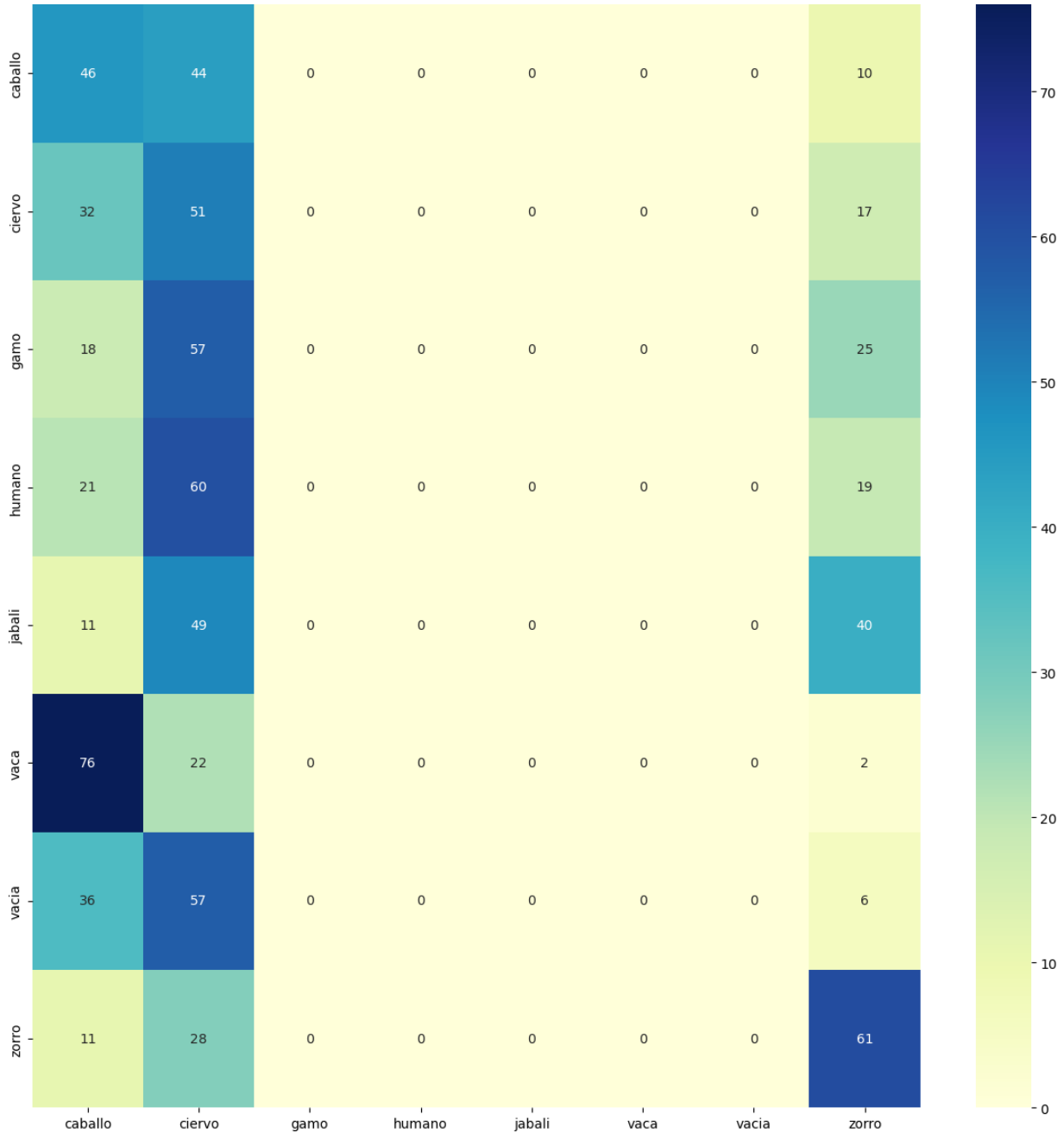

Figura 49: Matriz de confusión en la *Arquitectura de Crohn DROPOUT_FACTOR=0.2* sobre el conjunto de datos de test

Fuente: Elaboración propia

de los datos de validación, lo que indica que el problema subyacente de sobreaprendizaje no se ha resuelto.

El *accuracy* sobre el conjunto de datos de *test* fue extremadamente bajo. Esto sugiere que la red apenas aprendió a realizar predicciones significativas. De hecho, el rendimiento fue peor que en experimentos anteriores, lo que subraya la dificultad de entrenar modelos con este *dataset*.

Figura 50: Predicciones con la *Arquitectura de Crohn DROPOUT_FACTOR=0.2*

Fuente: Elaboración propia

La matriz de confusión reveló que las predicciones eran mucho peores que las que se obtendrían al azar. La mayoría de las clases se confundieron, lo que indica que el modelo tenía dificultades para distinguir entre las diferentes especies.

La confianza en las predicciones fue consistentemente baja tanto en las imágenes clasificadas correctamente como en las incorrectas. Esto sugiere que la red no estaba segura de sus propias predicciones, lo que contribuyó a la falta de precisión.

En conclusión, a pesar de los intentos por abordar el sobreaprendizaje mediante el ajuste del factor de *dropout*, el rendimiento de la red basada en la *Arquitectura de Crohn* no mejoró, sino que empeoró. Estos resultados destacan la complejidad del problema y sugieren que el *dataset* utilizado podría ser insuficiente o de mala calidad para entrenar una red neuronal con éxito. Además, la arquitectura de la red y los hiperparámetros pueden no estar optimizados para esta tarea específica. Se requerirían investigaciones adicionales y posiblemente la obtención de más datos de alta calidad para abordar eficazmente este desafío de clasificación de especies a través de imágenes de fototrampeo en el caso de poder resolverlo con este mismo *dataset*.

8.2. Análisis y discusión de resultados

El análisis y la discusión de los resultados obtenidos a lo largo de estos experimentos revelan un desafío significativo en la tarea de clasificación de especies a través de imágenes de fototrampeo. A pesar de los múltiples intentos de mitigar el sobreaprendizaje y mejorar el rendimiento de las redes neuronales, los resultados generales no han sido satisfactorios. En este apartado, se examinarán las razones detrás de estos resultados y se presentará una dirección para futuras investigaciones.

8.2.1. Problemas del Conjunto de Datos

A pesar de los desafíos identificados en los experimentos previos, es crucial reconocer la influencia de dos factores fundamentales en los resultados: la calidad del conjunto de datos y la implementación. La hipótesis principal que se postula es que los problemas radican en la calidad del conjunto de datos utilizado.

Uno de los aspectos críticos que ha contribuido a los resultados deficientes es la calidad y la cantidad limitada de datos disponibles en el conjunto de datos actual. Este conjunto de datos presenta varios problemas significativos que afectan su utilidad para la tarea de clasificación. Estos problemas incluyen:

- **Cantidad insuficiente de datos por clase**: La variabilidad de las imágenes en el conjunto de datos no se refleja adecuadamente debido a la falta de imágenes por clase. Esto limita la capacidad de las redes neuronales para aprender y generalizar patrones.
- **Objetos incompletos**: En muchas ocasiones, los objetos de interés (es decir, los animales) aparecen incompletos o parcialmente ocultos en las imágenes debido a la ubicación de las cámaras, el ángulo de captura o el movimiento de los propios animales. Esto hace que la tarea de reconocimiento sea aún más difícil, ya que los modelos deben aprender a reconocer a los animales incluso cuando no están completamente visibles.
- **Ráfagas de fotografías**: El proceso de fototrampeo a menudo resulta en ráfagas de fotografías en un corto período de tiempo, lo que significa que varias imágenes de una misma escena pueden caer en diferentes divisiones del conjunto de datos (entrenamiento, validación, prueba). Esto expone al modelo a imágenes casi idénticas durante el entrenamiento y validación y puede llevar a métricas poco fiables, pues si se le ha enseñado al modelo una imagen durante el entrenamiento seguramente sepa clasificarla bien durante la validación. Es decir, el *accuracy* de las redes presentadas podría ser incluso peor pues las métricas pueden no estar siendo precisas.
- **Ruido en los datos**: La presencia de imágenes de baja calidad, mal etiquetadas o con información irrelevante introduce ruido en el conjunto de datos, lo que dificulta que las redes neuronales aprendan patrones coherentes.

La mala calidad del conjunto de datos, junto con su insuficiente cantidad, ha sido un obstáculo significativo en los experimentos anteriores. Esto ha llevado a la formulación de la hipótesis de que los problemas observados no residen en las arquitecturas de las redes neuronales ni en los hiperparámetros utilizados, sino más bien en las limitaciones del conjunto de datos.

8.2.2. Próximos Pasos

Por otro lado, para descartar que no sea un fallo en la implementación, para determinar si estas arquitecturas de redes neuronales tienen el potencial de ofrecer buenos resultados en una tarea de clasificación, se llevará a cabo una evaluación adicional utilizando un conjunto de datos preparado específicamente para el estudio de la eficiencia de las arquitecturas.

En esta nueva fase de la investigación, se aplicarán las mismas arquitecturas de redes neuronales utilizadas en los experimentos anteriores a un conjunto de datos de alta calidad y diseñado para ser óptimo para la clasificación. El objetivo principal de este paso es aislar la influencia del conjunto de datos actual en los resultados y determinar si las arquitecturas de redes neuronales pueden, en sí mismas, lograr un alto rendimiento en una tarea de clasificación.

Es importante destacar que no se realizarán modificaciones en las arquitecturas de las redes ni en los hiperparámetros utilizados previamente. Esto permitirá una comparación directa con los resultados anteriores y ayudará a evaluar si el problema reside en el conjunto de datos o en la implementación.

La hipótesis detrás de este enfoque es que, con un conjunto de datos de alta calidad y una implementación adecuada, se pueden obtener resultados de precisión significativamente mejores. Este enfoque ayudará a determinar si las redes neuronales seleccionadas son adecuadas para la tarea de clasificación, o si los problemas anteriores se debieron principalmente a limitaciones en el conjunto de datos.

En conclusión, la próxima etapa de la investigación se centrará en la evaluación de las arquitecturas de redes neuronales en un contexto diferente, con un conjunto de datos de alta calidad y con la implementación. Esta evaluación proporcionará información crucial para comprender mejor las capacidades de las redes y la naturaleza de los desafíos enfrentados en los experimentos anteriores.

9. Aplicación a un *dataset* de entorno controlado

9.1. Introducción

Tras llevar a cabo los experimentos sobre el *dataset* de imágenes de fototrampeo, se ha observado que los resultados no han sido satisfactorios debido a la complejidad y la insuficiente cantidad de datos disponibles en dicho *dataset*. Las imágenes provenientes de fototrampeo presentan una serie de desafíos adicionales, como la variabilidad en las condiciones de iluminación, el enfoque variable de los animales y la presencia de vegetación que puede ocultar a los sujetos de interés entre otos . Estas dificultades han llevado al sobreajuste y a una baja capacidad de generalización de los modelos.

La limitación en la cantidad de datos de fototrampeo etiquetados y la presencia de ruido y ambigüedad en el *dataset* hacen que sea difícil para los modelos de *deep learning* generalizar adecuadamente a nuevas imágenes de fototrampeo. Por lo tanto, es esencial explorar y evaluar el rendimiento de los modelos en un entorno controlado, donde se pueda obtener un conjunto de datos más completo y etiquetado de manera precisa.

Con el objetivo de valorar si el problema radica en la implementación o en la naturaleza del *dataset*, se procederá a realizar experimentos sobre un *dataset* de entorno controlado y adecuado para la clasificación de imágenes. Este nuevo conjunto de datos debe haber sido cuidadosamente preparado y etiquetado para garantizar una mejor calidad y cantidad de información, evitando los problemas de ruido y ambigüedad presentes en el *dataset* de fototrampeo.

Al realizar los experimentos sobre el *dataset* de entorno controlado, se espera lograr los siguientes resultados:

- **Mejor rendimiento del modelo**: Dado que el *dataset* de entorno controlado ha sido preparado para reducir el ruido y la ambigüedad, se espera obtener un rendimiento mejorado del modelo en comparación con los resultados previos en el *dataset* de fototrampeo.
- **Mayor capacidad de generalización**: La naturaleza más controlada del *dataset* permitirá que el modelo generalice mejor a nuevas imágenes, superando el problema de sobreajuste observado en el *dataset* de fototrampeo.
- **Identificación de la importancia del *dataset***: Al contrastar los resultados entre ambos *dataset*s, se podrá demostrar que los problemas encontrados en el *dataset* de fototrampeo no se deben a fallos de implementación, sino a la complejidad y falta de información de dicho *dataset*.
- **Validación del enfoque de *deep learning***: Los resultados exitosos en el *dataset* de entorno controlado validarán el enfoque de *deep learning* en la clasificación de imágenes y su potencial para tareas más complejas.

Con estos resultados, se reforzará la importancia de utilizar conjuntos de datos adecuados y de calidad para el entrenamiento de modelos de *deep learning*. Además, se proporcionará una sólida base para la discusión y las conclusiones finales del trabajo, resaltando la relevancia de la selección adecuada de *dataset*s en el campo de la inteligencia artificial y la visión por computador.

9.2. Base de datos

El *dataset* utilizado en este conjunto de experimentos se ha conseguido de Kaggle [59]. Kaggle es una plataforma en línea que alberga una comunidad de científicos de datos, entusiastas del aprendizaje automático y analistas que participan en competiciones y colaboran en proyectos relacionados con la ciencia de datos. La plataforma ofrece una amplia variedad de conjuntos de datos públicos y privados, así como herramientas y recursos para el desarrollo y evaluación de modelos de aprendizaje automático. Para este trabajo, la base de datos para el *dataset* de entorno controlado se ha buscado específicamente en Kaggle, seleccionando un conjunto de imágenes de verduras que ha sido ampliamente utilizado en desafíos de clasificación de imágenes para garantizar la calidad y diversidad necesaria en el proceso de experimentación.

El conjunto de datos *Vegetable Image dataset* [60] consiste en una colección de imágenes de vegetales, específicamente de diferentes tipos de verduras. La elección de este *dataset* se debe a su calidad, variedad y cantidad de imágenes disponibles, lo que permitirá una mejor evaluación de los modelos utilizados.

El conjunto de datos se compone de imágenes en color, en su mayoría centradas y bien enfocadas, lo que debería facilitar el proceso de clasificación. Además, las imágenes han sido cuidadosamente etiquetadas y organizadas en categorías que corresponden a diferentes tipos de verduras.

9.2.1. Conjunto de Datos

El conjunto de datos utilizado para la evaluación de la eficiencia del modelo sobre conjuntos de datos de un entorno controlado para el entrenamiento de redes neuronales es un conjunto de datos de 15 clases de verduras. Estas clases son *Bean*, *Bitter Gourd*, *Bottle Gourd*, *Brinjal*, *Broccoli*, *Cabbage*, *Capsicum*, *Carrot*, *Cauliflower*, *Cucumber*, *Papaya*, *Potato*, *Pumpkin*, *Radish*, *Tomato*, en español: Judía, Calabaza Amarga, Calabaza de Botella, Berenjena, Brócoli, Repollo, Pimiento, Zanahoria, Coliflor, Pepino, Papaya, Patata, Calabaza, Rábano, Tomate. Este conjunto de datos se ha obtenido de *Kaggle*, una plataforma en línea que permite acceder a conjuntos de datos, participar en desafíos de análisis y modelado de datos, y colaborar con otros profesionales en proyectos de ciencia de datos [61]. En la Figura 51 se muestra una imagen por clase del conjunto de entrenamiento.

9.2.2. Conjuntos de entrenamiento, validación y test

La selección de las imágenes para los conjuntos de entrenamiento, validación y test ya se había realizado, siendo aproximadamente la proporción de 70 %, 15 % y 15 %, respectivamente, del total de 1400 imágenes por clase que contiene el *dataset*. Esto significa que el 70 % de las imágenes serán utilizadas para entrenar los modelos, mientras que el 15 % se reservará para la validación y el 15 % restante para las pruebas. La distribución de las clases se muestra en la Figura 52.

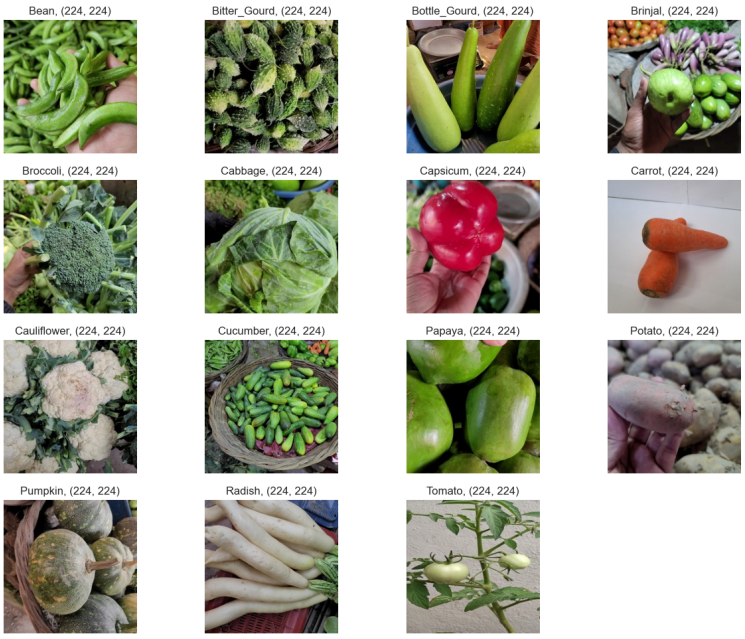

Figura 51: Muestra de imágenes del *dataset* de verduras

Fuente: Elaboración propia

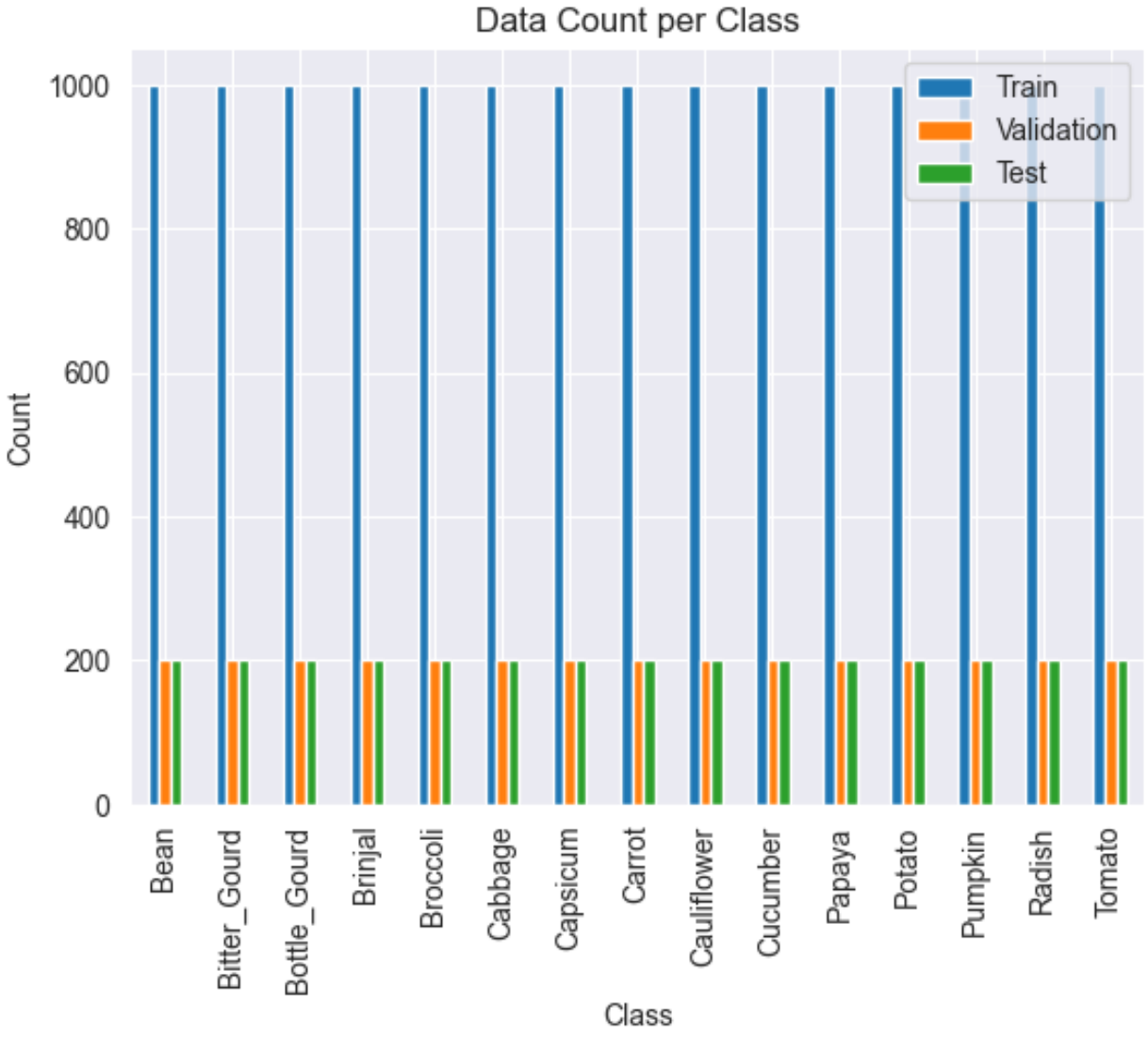

Figura 52: Distribución de las clases

Fuente: Elaboración propia

9.2.3. *Data Augmentation*

En la Figura 53, se muestran algunas imágenes generadas mediante el *Data Augmentation* correspondientes a los conjuntos de datos antes presentados. Estas imágenes representan variaciones de las imágenes originales y proporcionan al modelo una mayor cantidad y diversidad de datos de entrenamiento.

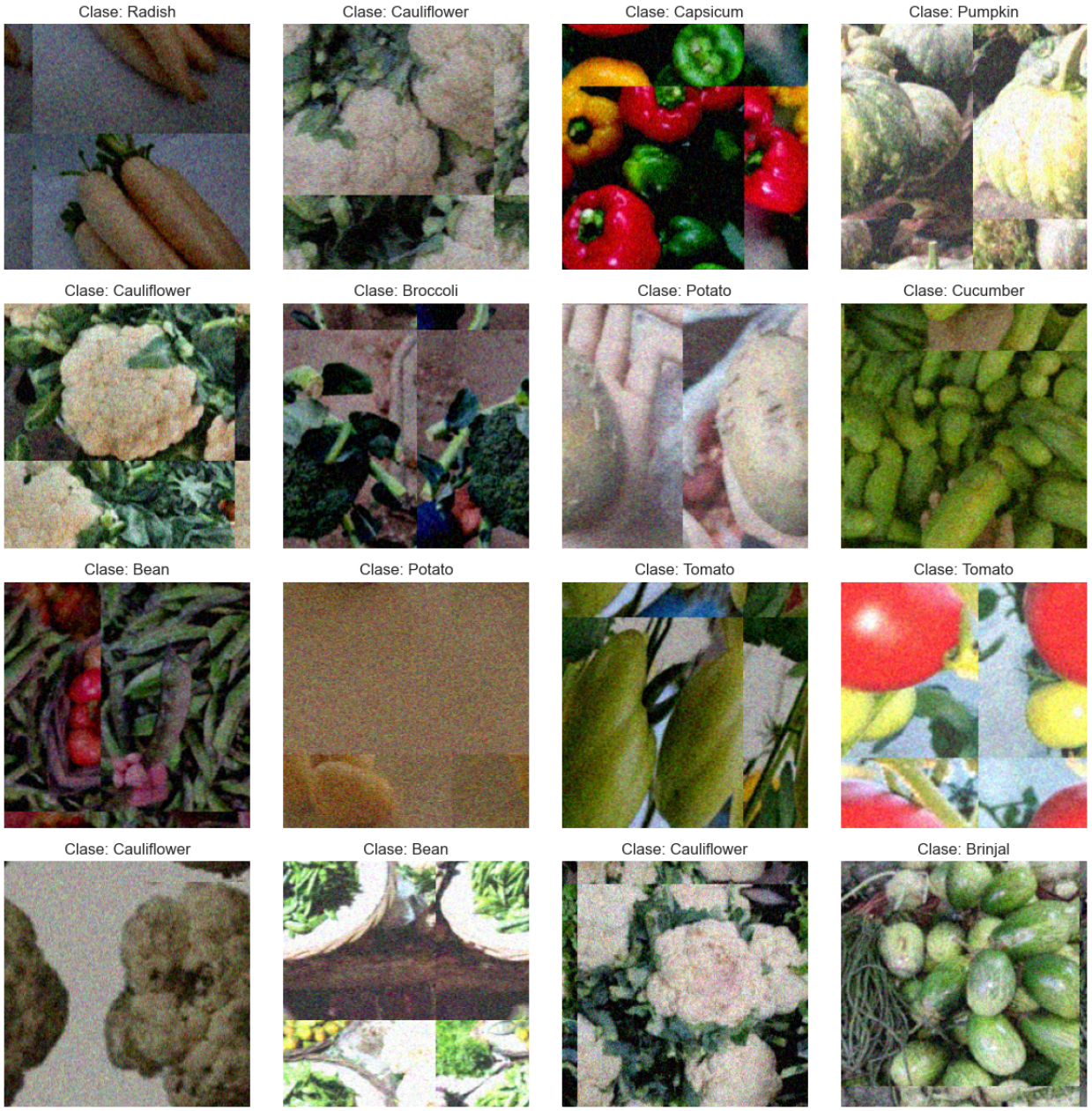

Figura 53: Muestra del conjunto de datos de verduras con *Data Augmentation*

Fuente: Elaboración propia

El mismo conjunto de técnicas de *Data Augmentation* utilizado en el conjunto de datos anterior se ha aplicado aquí. Sin embargo, lo más destacado sigue siendo la técnica de *random translation*. Aunque en las imágenes también se pueden observar otros efectos, como ruido o aumentos en el brillo y la saturación, el *random translation* sigue siendo la técnica más notable.

9.3. Metodología

La metodología seguida para los experimentos con el conjunto de datos de verduras será esencialmente la misma que la aplicada a las imágenes de fototrampeo. Esto implica que se utilizarán las mismas arquitecturas de redes neuronales convolucionales, se ajustarán los mismos hiperparámetros, se aplicarán los mismos *callbacks* y se mantendrán las mismas técnicas de entrenamiento, incluido el uso de *Transfer Learning* cuando sea posible y *Data Augmentation*.

El objetivo es establecer una comparación directa entre el rendimiento de los modelos en el conjunto de datos de entorno controlado y los resultados obtenidos previamente en el conjunto de datos de fototrampeo. Esto permitirá evaluar si los problemas identificados en los experimentos anteriores se deben principalmente a la naturaleza del conjunto de datos o si también están relacionados con la propia implementación o la elección de arquitecturas e hiperparámetros.

En resumen, se realizarán los mismos tipos de experimentos y análisis que se llevaron a cabo para el conjunto de datos de animales en el entorno de fototrampeo, pero esta vez aplicados al conjunto de datos de verduras pertenecientes a un entorno controlado. Esto permitirá determinar si los modelos seleccionados son adecuados para la tarea de clasificación de imágenes en general y si los problemas anteriores se deben principalmente a limitaciones en el conjunto de datos de fototrampeo.

9.4. Resultados

A continuación, se presentarán y discutirán los resultados obtenidos a partir de los experimentos realizados en el conjunto de datos de verduras, un entorno controlado y adecuado para la clasificación de imágenes. Estos experimentos tienen como objetivo evaluar el rendimiento de las arquitecturas de redes neuronales y las estrategias de entrenamiento previamente seleccionadas en condiciones ideales, en contraste con los desafíos encontrados en el conjunto de datos de fototrampeo.

9.4.1. MobileNetV2 ($DROPOUT = 0.2$)

La evolución del entrenamiento que se muestra en la Figura 54, muestra el progreso de la red *MobileNetV2* durante las diferentes épocas de entrenamiento. Se puede apreciar que el accuracy aumenta gradualmente, alcanzando un valor de 99,93 % en la última época. El loss, por otro lado, disminuye a medida que avanza el entrenamiento, llegando a un valor de 0.000240 en la última época. Aunque el modelo guardado es el que corresponde a un mejor accuracy sobre el conjunto de datos de validación, en el que se consiguió un 99,1 % de accuracy.

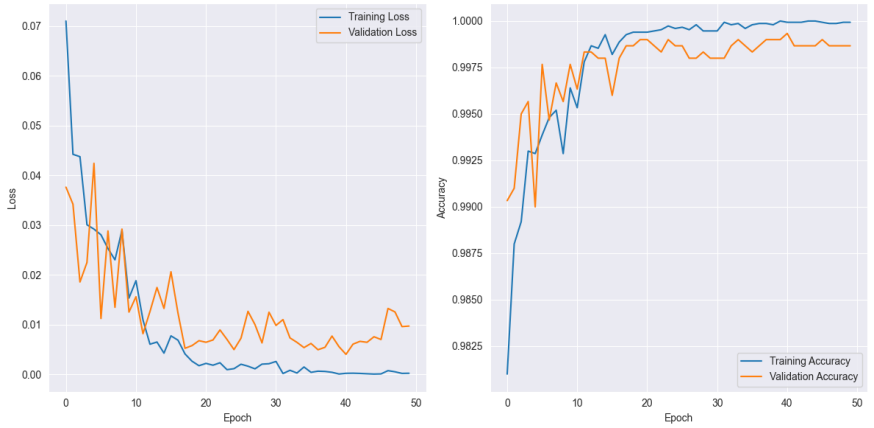

Figura 54: Evolución del entrenamiento *frozen* en MobileNetV2 *DROPOUT_FACTOR=0.2*

Fuente: Elaboración propia

Al ejecutar el modelo con el mejor accuracy sobre los datos de validación sobre el conjunto de datos de test se obtuvo un accuracy de 99,2 % y un loss de 0.0376. Lo que muestra que el modelo reconoce las clases en imágenes no antes vistas de forma bastante precisa.

En la Figura 55, se presenta la matriz de confusión obtenida después del entrenamiento *frozen*. Se puede observar que la mayoría de las clases tienen una alta precisión, ya que la diagonal principal de la matriz está compuesta por valores cercanos al total de imágenes de cada clase en el conjunto de test, lo cual indica una correcta clasificación. Sin embargo, se observa una confusión entre las clases *cucumber* y *brinjal*, aunque el número de clasificaciones erróneas es insignificante en comparación con el número de aciertos en la mayoría de las clases. Además, hay que tener en cuenta que el modelo no ha sido entrenado en su totalidad, pues en la red *MobileNetV2* se llevan a cabo dos entrenamientos, el primero sólo sobre las capas superiores.

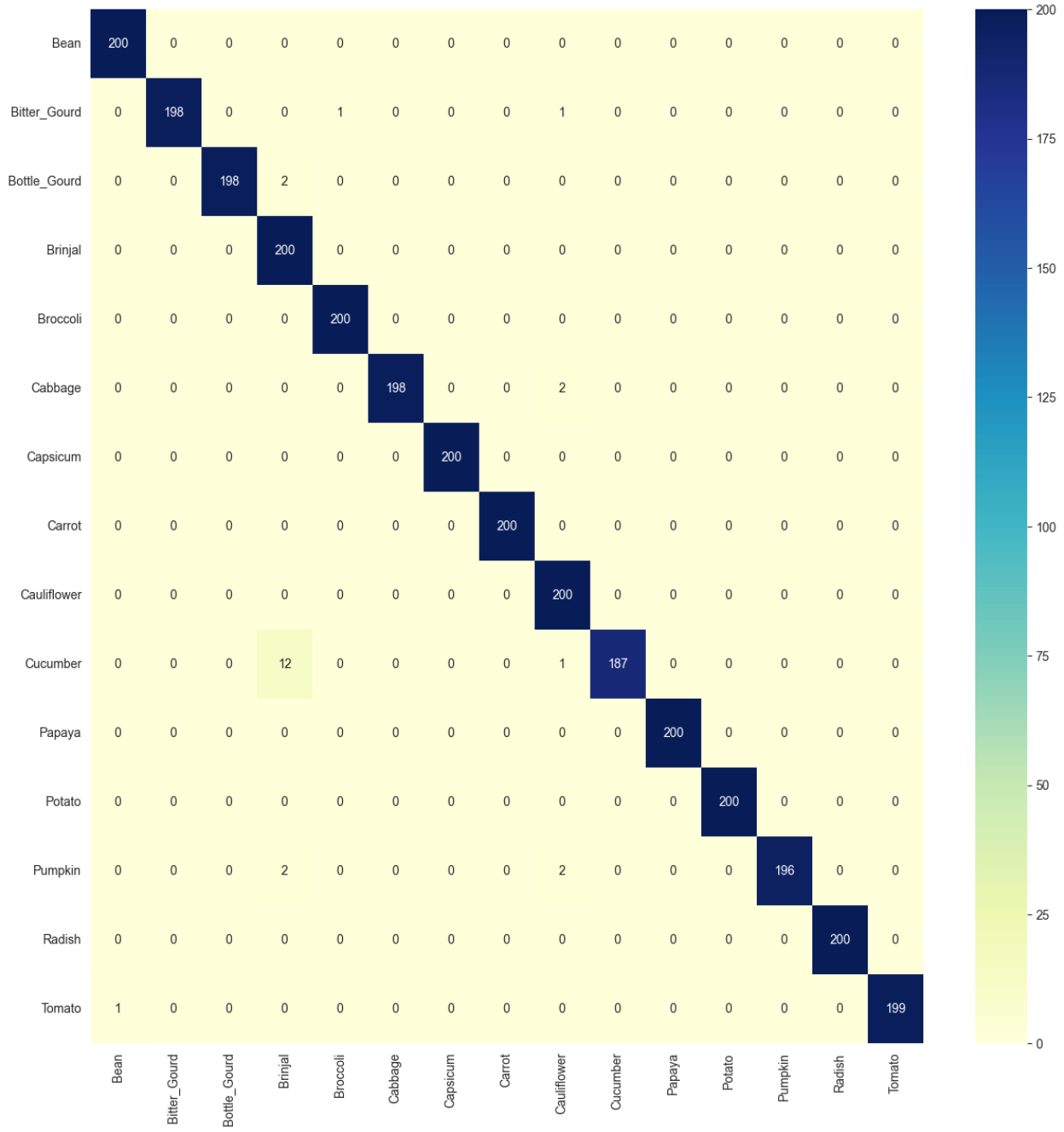

Figura 55: Matriz de confusión tras el entrenamiento *frozen* en MobileNetV2 *DROPOUT_FACTOR=0.2* sobre el conjunto de datos de test

Fuente: Elaboración propia

Después del entrenamiento *unfrozen*, cuya evolución puede verse en la Figura 56 se realiza sobre todas las capas del modelo. El margen de mejora de tanto loss y accuracy es pequeño pero aún así se ha conseguido mejorar.

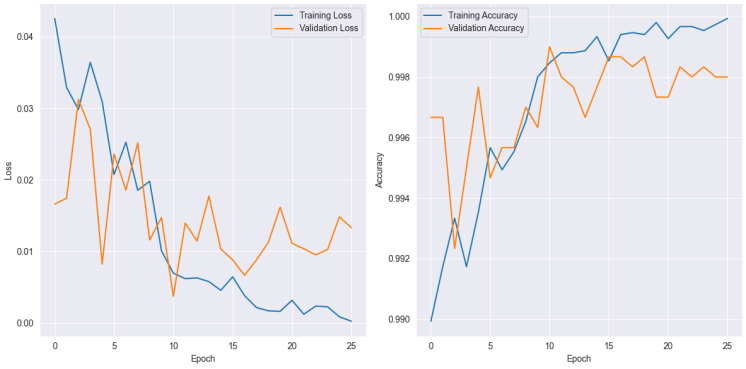

Figura 56: Evolución del entrenamiento *unfrozen* en *MobileNetV2 DROPOUT_FACTOR=0.2*

Fuente: Elaboración propia

Además, se logra resolver la confusión que se obtenía entre las clases *cucumber* y *brinjal* y se obtiene una matriz de confusión casi perfecta, esta matriz se muestra en la Figura 57. En esta matriz, se puede apreciar que la diagonal principal está compuesta por valores cercanos a 200, lo cual indica una alta precisión en la clasificación de las clases.

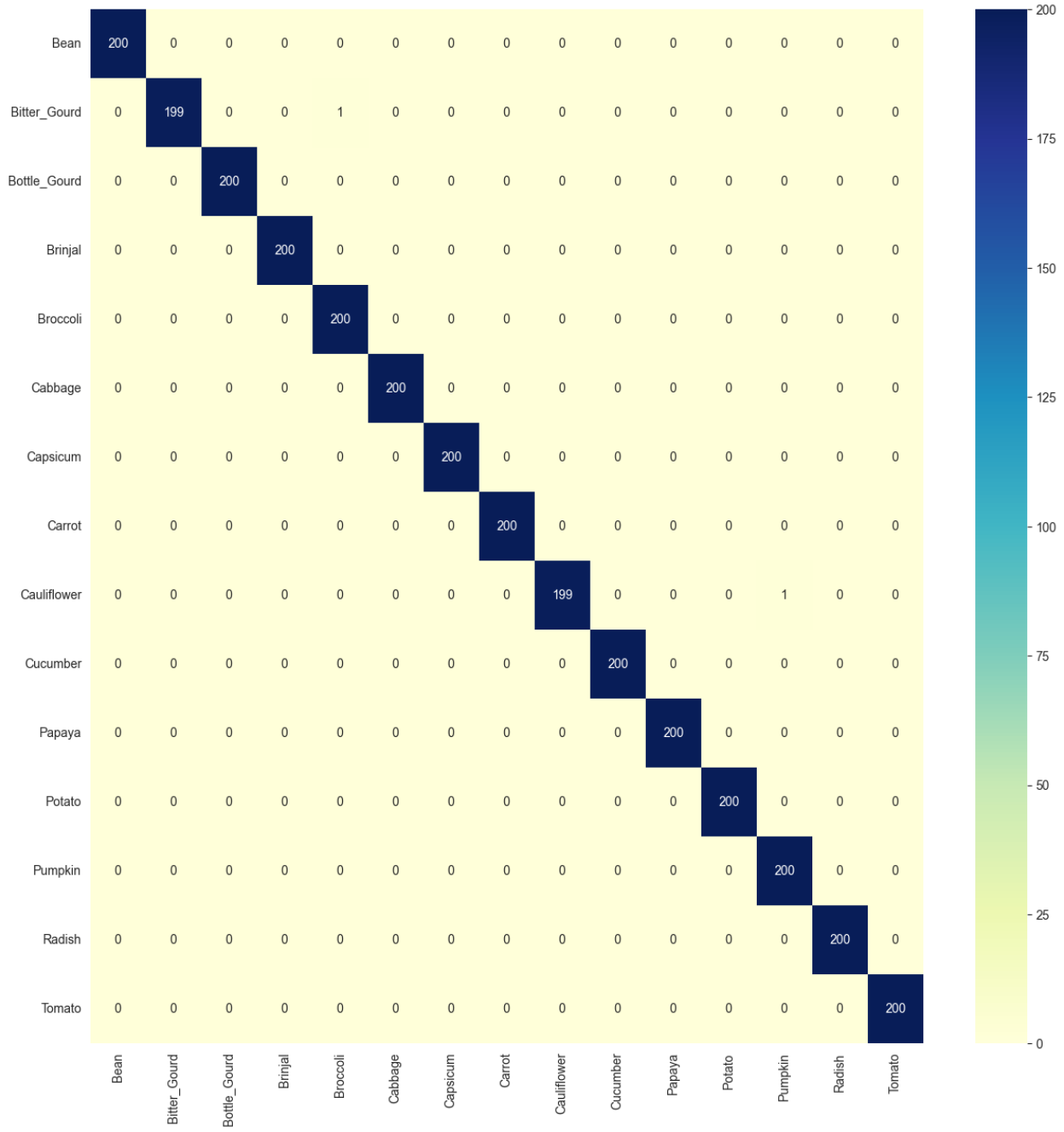

Figura 57: Matriz de confusión tras el entrenamiento *unfrozen* en *MobileNetV2 DROPOUT_FACTOR=0.2* sobre el conjunto de datos de test

Fuente: Elaboración propia

En la Figura 58, se presentan algunas predicciones realizadas por la red *MobileNetV2* sobre el conjunto de datos de prueba. Estas predicciones muestran la capacidad de la red para clasificar correctamente diferentes tipos de verduras. Las imágenes mostradas han sido seleccionadas de forma aleatoria, y todas las que se han mostrado han sido clasificadas correctamente por la red, con un 100 % de accuracy en la predicción de la clase.

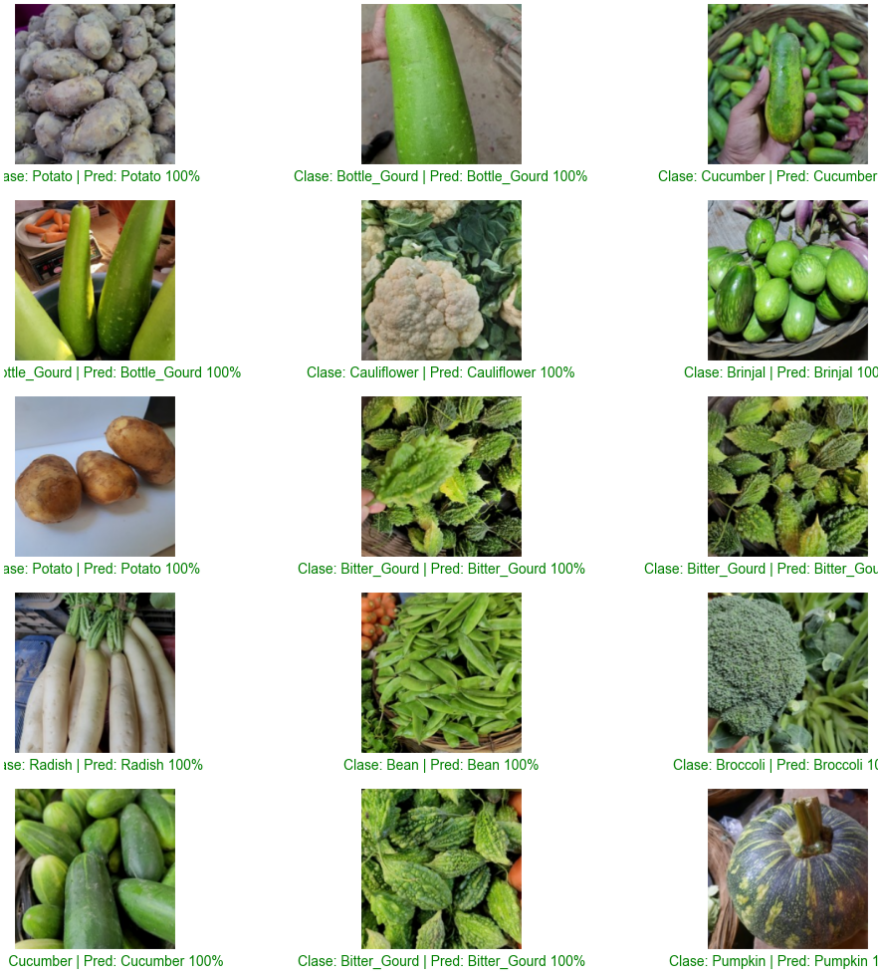

Figura 58: Predicciones con *MobileNetV2*

Fuente: Elaboración propia

En resumen, los resultados obtenidos muestran que la red *MobileNetV2* entrenada sobre el *dataset* de verduras logra altos niveles de precisión en la clasificación. La evolución del entrenamiento demuestra un aumento gradual en el accuracy y una disminución del loss a medida que avanza el entrenamiento, tanto sobre los datos de entrenamiento como sobre los de validación. Y posteriormente obteniendo los mismos resultados sobre los datos de test. Además, las matrices de confusión muestran una clasificación casi perfecta, con una ligera confusión inicial entre las clases *cucumber* y *brinjal* que se soluciona en el entrenamiento *unfrozen*. Esto indica que la red es capaz de distinguir eficientemente entre las diferentes clases de verduras en el *dataset* utilizado.

9.4.2. *Arquitectura de Crohn*

La evolución del entrenamiento que se muestra en la Figura 59, muestra el progreso de la *Arquitectura de Crohn* durante las diferentes épocas de entrenamiento. Se puede apreciar que el accuracy aumenta gradualmente, alcanzando un valor de 98,35 % en la última época. El loss, por otro lado, disminuye a medida que avanza el entrenamiento, llegando a un valor de 0.0504 en la última época. Aunque el modelo guardado es el que corresponde a un mejor accuracy sobre el conjunto de datos de validación, en el que se consiguió un 98,87 % de accuracy. Es importante tener en cuenta que en esta red el tiempo de entrenamiento es mucho mayor que en la de *MobileNetV2*, en la*Arquitectura de Crohn* el tiempo de entrenamiento para una epoch es de 2500 segundos aproximadamente mientras que para la red *MobileNetV2* es de 250 segundos para cada época. Por esto, el entrenamiento sobre la *Arquitectura de Crohn* se ha reducido en cuanto al número de épocas.

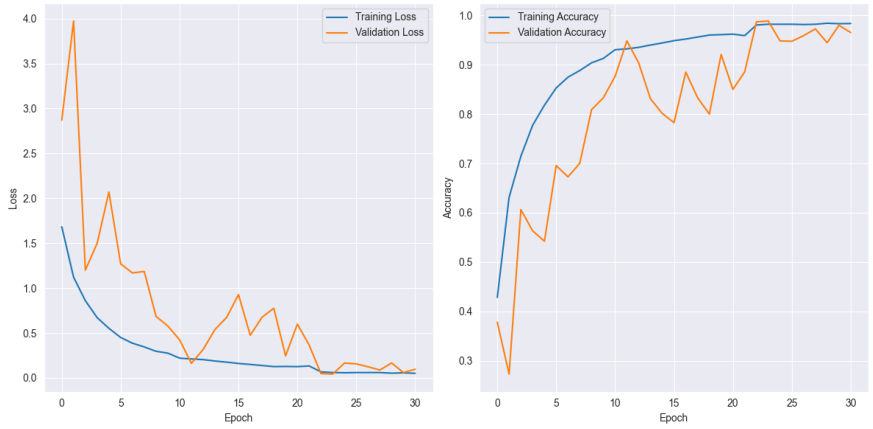

Figura 59: Evolución del entrenamiento en la *Arquitectura de Crohn sin DROPOUT*

Fuente: Elaboración propia

Al ejecutar el modelo con el mejor accuracy sobre los datos de validación sobre el conjunto de datos de test se obtuvo un accuracy de 98,8 % y un loss de 0.0437. Lo que muestra que el modelo reconoce las clases en imágenes no antes vistas de forma bastante precisa aunque no tan bien como el modelo anterior.

En la Figura 60, se presenta la matriz de confusión obtenida después del único entrenamiento que se realiza sobre este modelo. Se puede observar que la mayoría de las clases tienen una alta precisión, ya que la diagonal principal de la matriz está compuesta por valores cercanos al total de imágenes de cada clase en el conjunto de test, lo cual indica una correcta clasificación. No hay clases que confunda de forma común, aunque sí que hay algunas imágenes que no clasifica correctamente, pero el número de imágenes mal clasificadas es muy pequeño frente al número de imágenes clasificadas correctamente.

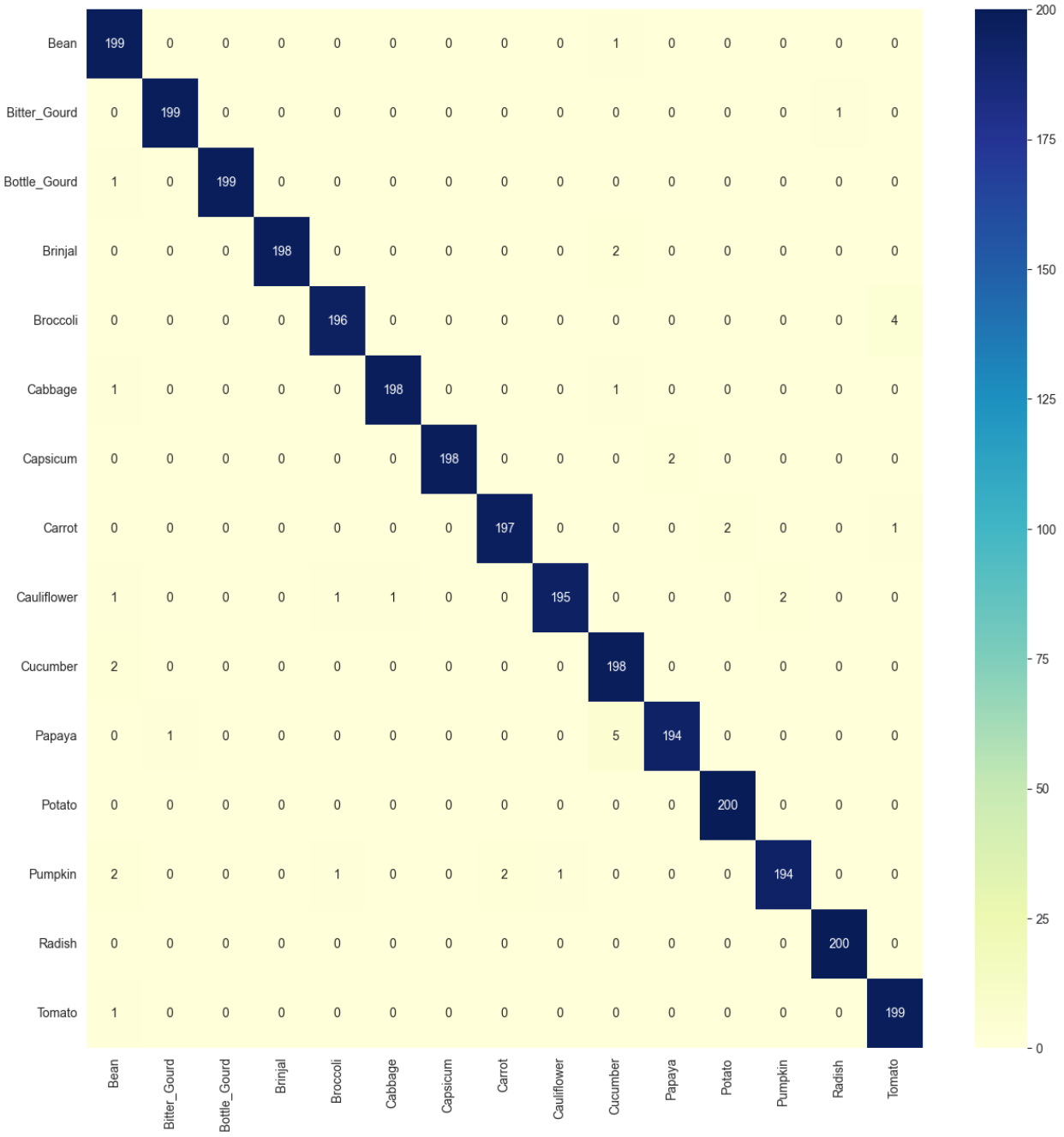

Figura 60: Matriz de confusión en la *Arquitectura de Crohn sin DROPOUT* sobre el conjunto de datos de test

Fuente: Elaboración propia

En la Figura 61, se presentan algunas predicciones realizadas por la *Arquitectura de Crohn* sobre el conjunto de datos de prueba. Estas predicciones muestran la capacidad de la red para clasificar correctamente diferentes tipos de verduras. Las imágenes mostradas han sido seleccionadas de forma aleatoria, todas las que se han mostrado y han sido clasificadas correctamente por la red, ha sido con un 100 % de accuracy en la predicción de la clase. Sin embargo, hay una imagen de las mostradas que no ha sido clasificada correctamente, ha confundido la clase *papaya* clasificandola como *Cucumber* con un accuracy de 91 %.

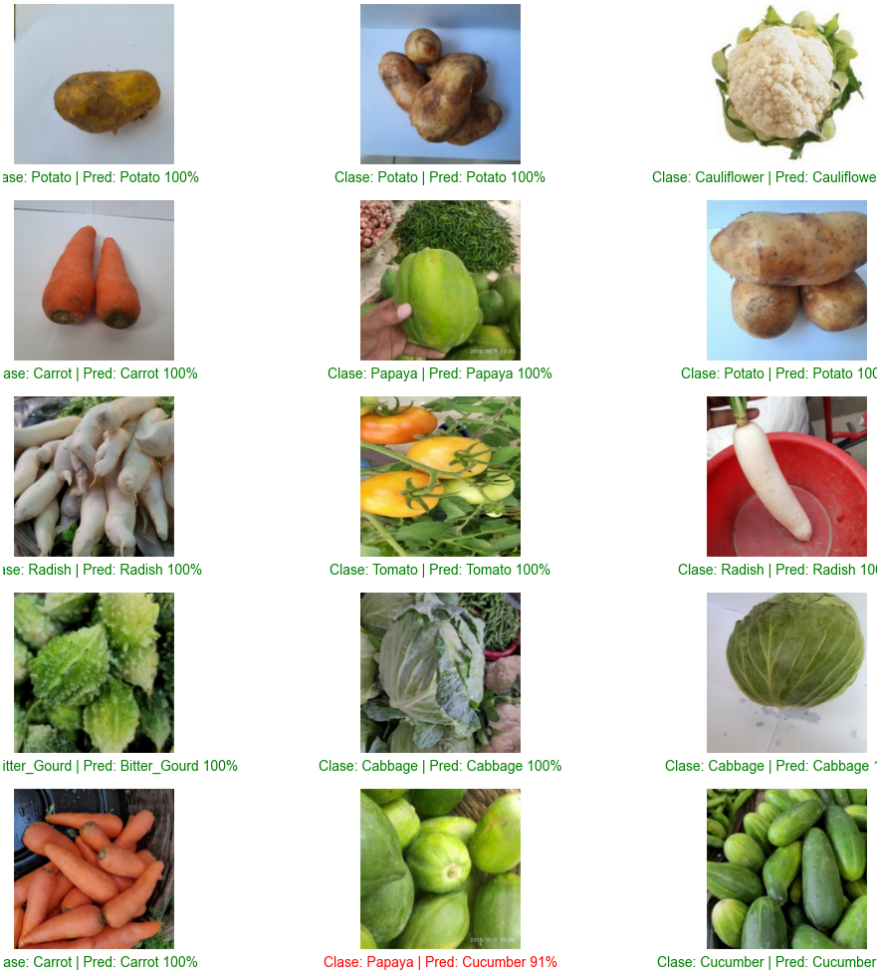

Figura 61: Predicciones con la *Arquitectura de Crohn*

Fuente: Elaboración propia

En resumen, los resultados obtenidos muestran que la *Arquitectura de Crohn* entrenada sobre el *dataset* de verduras logra altos niveles de precisión en la clasificación. La evolución del entrenamiento demuestra un aumento gradual en el *accuracy* y una disminución del *loss* a medida que avanza el entrenamiento, tanto sobre los datos de entrenamiento como sobre los de validación. Y posteriormente obteniendo los mismos resultados sobre los datos de test. Además, la matriz de confusión muestran una clasificación casi perfecta. Esto indica que la red es capaz de distinguir eficientemente entre las diferentes clases de verduras en el *dataset* utilizado.

Es importante tener en cuenta el tiempo de entrenamiento requerido para entrenar a esta red y que en este caso no se ha realizado *Transfer Learning*. Los resultados obtenidos han sido muy buenos pero ha requerido mucho más tiempo de entrenamiento consiguiendo un accuracy inferior.

9.5. Análisis

Los resultados obtenidos en los experimentos con el conjunto de datos de verduras, en comparación con los resultados previos en el conjunto de datos de animales, son notables y reveladores.

MobileNetV2 entrenada con un factor de dropout de 0,2 muestra un rendimiento excepcional. Al observar la evolución del entrenamiento, se ve claramente cómo el modelo aprende de manera constante a medida que avanzan las épocas. El modelo logra un accuracy impresionante de alrededor del 99.2 % en el conjunto de datos de *test*, lo que indica su capacidad para generalizar y clasificar con alta precisión imágenes de verduras nunca antes vistas. Las matrices de confusión revelan una confusión inicial mínima entre dos clases, que se resuelve en el entrenamiento *unfrozen*, lo que refuerza aún más la capacidad del modelo para aprender y adaptarse. Además, las predicciones muestran que el modelo clasifica correctamente las imágenes de verduras con una precisión del 100 % para cada clasificación.

Por otro lado, la *Arquitectura de Crohn*, sin el uso de dropout y sin transfer learning, también muestra un rendimiento notable. Aunque el tiempo de entrenamiento es significativamente mayor en comparación con *MobileNetV2*, la red logra un *accuracy* de alrededor del 98.8 % en el conjunto de datos de *test*. Las matrices de confusión indican una clasificación efectiva de las clases de verduras, con un mínimo número de imágenes mal clasificadas. Sin embargo, se observa una confusión ocasional.

En resumen, estos resultados sugieren que las arquitecturas de redes neuronales y las estrategias de entrenamiento seleccionadas son altamente efectivas para la clasificación de imágenes, al menos en un entorno controlado y con un conjunto de datos de alta calidad como el de verduras. Esto respalda la hipótesis de que los problemas encontrados en el conjunto de datos de fototrampeo no se debían a fallos de implementación, sino más bien a la complejidad y la calidad insuficiente del conjunto de datos. Los modelos demuestran una capacidad significativa para aprender y generalizar, lo que enfatiza la importancia de contar con conjuntos de datos adecuados y de calidad para el entrenamiento de modelos de *deep learning*. Además, estos resultados proporcionan una sólida base para la discusión y las conclusiones finales del trabajo, destacando la relevancia de la selección adecuada de conjuntos de datos en el campo de la inteligencia artificial y la visión por computador.

10. Conclusiones

A lo largo de este estudio, se ha explorado el campo de intersección de la ecología y la inteligencia artificial. El objetivo central ha sido desarrollar modelos de *deep learning* capaces de abordar una tarea fundamental en ecología: la clasificación de especies a partir de imágenes de fototrampeo. Sin embargo, en el proceso se han afrontado desafíos y descubrimientos importantes.

A continuación, se presentan las conclusiones finales de este trabajo de investigación, que abarcan aspectos técnicos, consideraciones sobre trabajos futuros y reflexiones personales sobre el proceso de desarrollo y aprendizaje a lo largo del proyecto.

10.1. Conclusiones técnicas

Para abordar el desafío de clasificar especies de animales en imágenes de fototrampeo, se ha explorado la integración de cámaras de fototrampeo y técnicas avanzadas de visión por computador, específicamente las CNN. Esta combinación tiene el potencial de revolucionar el monitoreo de la biodiversidad y la conservación de especies al proporcionar una solución automatizada y eficiente.

El conjunto de datos utilizado en este estudio, extraído de la base de datos de imágenes de fototrampeo del proyecto AI-Census de la Universidad de Huelva [57], ha sido fundamental para la experimentación y evaluación de los modelos de clasificación de especies. Con aproximadamente 3.5 millones de imágenes etiquetadas y una distribución desbalanceada de 21 clases de especies diferentes, este conjunto de datos ofrece una representación diversa y realista de los escenarios de fototrampeo.

Sin embargo, en la experimentación se utilizó un subconjunto de este conjunto de datos, que contiene muchas menos imágenes y anotaciones. Se seleccionaron 8000 imágenes de las 8 clases con más imágenes, sin tener en cuenta su ubicación ni fecha. Esto influyó en la creación de los conjuntos de entrenamiento, validación y *test*. Las imágenes presentan complejidades, ya que los animales pueden aparecer en diferentes ubicaciones de la imagen, y la falta de datos de ubicación y fecha, así como la falta de información sobre la ráfaga de imágenes a la que pertenecen, impiden asegurar la independencia de estos conjuntos. Esto afecta la fiabilidad de los valores obtenidos en las métricas utilizadas, como el *accuracy*, ya que imágenes de la misma ráfaga pueden estar presentes tanto en entrenamiento como en validación, lo que podría llevar a una evaluación optimista del modelo. Las fechas y ubicaciones podrían haber sido factores útiles, ya que podrían proporcionar información sobre las condiciones temporales y ambientales pues afecta directamente a las imágenes. En el caso temporal afecta al fondo de las imágenes que pertenecen al entorno natural de Doñana, este lugar puede verse muy cambiado en una misma ubicación entre dos épocas del año. Por otro lado la hora podría haber ayudado a buscar un equilibrio entre imágenes nocturnas y diurnas.

Otro desafío importante es la escasez de imágenes disponibles para un problema tan complejo. Con solo 1000 imágenes para 8 clases, muchas imágenes son muy similares debido a la ubicación y la dificultad de los animales en las imágenes. Aumentar significativamente el número de imágenes por clase sería fundamental para mejorar la capacidad de generalización del modelo.

A medida que avanzaba el trabajo, se hicieron varias observaciones importantes:

- La experimentación reveló que los cambios en los hiperparámetros no condujeron a mejoras sustanciales en la eficiencia de la red.

- La calidad y diversidad del conjunto de datos resultaron críticas para el rendimiento del modelo. Se trato de mejorar con la inclusión de técnicas de *data augmentation* y la optimización del *dropout* pero no fueron tan efectivas y en algunos casos llegaron a ser contraproducentes.
- Se destacó la importancia de la independencia entre conjuntos de entrenamiento y validación. La falta de independencia entre los datos puede llevar a resultados optimistas y afectar la capacidad de generalización del modelo.
- Se reconoció que los desafíos específicos de la ecología, como las ráfagas de imágenes similares, deben abordarse cuidadosamente al interpretar los resultados.

En resumen, para abordar con éxito el desafío de la clasificación de especies a través de imágenes de fototrampeo, es esencial dedicar tiempo a la elaboración de un conjunto de datos diverso y de alta calidad. El *data augmentation* y la optimización de los hiperparámetros pueden ayudar, pero la base del éxito está en la preparación adecuada de los datos. Esto permitirá obtener resultados más sólidos y permitirá centrarse en otros aspectos, como la optimización de la red.

10.2. Conclusiones personales

La aplicación de técnicas de visión por computador en ecología y conservación representa un avance en la forma en que comprendemos y protegemos la biodiversidad. Esta tecnología tiene el potencial de transformar la recopilación de datos y el monitoreo de especies, al tiempo que reduce el disturbio a la fauna y la dependencia de métodos intrusivos.

Durante la realización de este estudio he llegado a apreciar la importancia de adoptar enfoques interdisciplinarios que combinen la biología y la informática. La colaboración entre expertos en ecología y científicos de datos es esencial para desarrollar soluciones efectivas y éticas para los desafíos de la conservación. Además, este trabajo ha destacado la importancia de abordar la calidad y diversidad de los conjuntos de datos utilizados en la clasificación de especies, ya que esto puede tener un impacto significativo en los resultados.

Una de las lecciones más valiosas que he aprendido a lo largo de este proyecto es la importancia de considerar incluso los detalles del conjunto de datos que aparentemente no tienen relación con el objetivo final. Las secuencias de imágenes y su distribución en conjuntos de entrenamiento, validación y *test* pueden tener un impacto significativo en los resultados. La posibilidad de que imágenes de la misma secuencia caigan en conjuntos diferentes puede llevar a una evaluación optimista del modelo. Esta es una consideración crítica en la ecología y otras áreas de estudio que utilizan imágenes de fototrampeo. Esta experiencia me ha proporcionado una intuición valiosa sobre qué aspectos tener en cuenta en futuros proyectos de procesamiento de imágenes.

Además, he llegado a entender la importancia de la salida de campo en el proceso de investigación. La oportunidad de participar en salidas de campo para recoger imágenes directamente ha sido esencial para comprender todo el proceso, desde la recopilación de datos hasta el procesamiento. Esto me ha permitido adquirir una comprensión completa y contextualizada de cada etapa del estudio y apreciar la importancia de cada una de ellas.

También ha sido enriquecedor descubrir las similitudes y conexiones entre áreas aparentemente muy diferentes como la informática y la ecología. El contacto con investigadores de diferentes campos me ha enseñado la importancia de la colaboración interdisciplinaria y me ha brindado la capacidad de participar de manera efectiva en entornos de trabajo diversos. Esta experiencia me

ha proporcionado una visión más amplia y una mayor apreciación de las posibilidades y desafíos que enfrenta la investigación interdisciplinaria.

En resumen, este proyecto ha sido una experiencia valiosa que ha enriquecido mi comprensión de la ciencia, la tecnología y la colaboración interdisciplinaria.

También ha quedado claro que el éxito en la implementación de modelos de *deep learning* está intrínsecamente ligado a la calidad de los datos de entrenamiento y a la elección adecuada de técnicas de preprocesamiento. El *Data Augmentation* y la adaptación de factores de *Dropout* son herramientas poderosas para mejorar la eficacia de los modelos en situaciones en las que los datos son limitados o ruidosos aunque cuando la limitación es muy grande puede ser incluso contraproducentes.

En resumen, este proyecto ha demostrado que la intersección de la ecología y la inteligencia artificial tiene un potencial significativo para abordar desafíos en la conservación y el monitoreo de la biodiversidad. Las lecciones aprendidas y los resultados obtenidos proporcionan una base sólida para futuras investigaciones y aplicaciones en este campo de estudio.

10.3. Trabajos Futuros

Para futuras investigaciones relacionadas con este estudio, existen varias oportunidades de mejora y expansión que pueden contribuir significativamente a la clasificación de especies en imágenes de fototrampeo.

Uno de los aspectos cruciales que podría mejorar la calidad de los resultados es la ampliación del conjunto de datos. Como se mencionó anteriormente, el estudio se basó en un subconjunto relativamente pequeño de imágenes. Para lograr un modelo más robusto y generalizable, se puede considerar la utilización de un conjunto de datos más grande y diverso. Esto podría incluir la inclusión de más clases de especies y un mayor número de imágenes por clase.

Además, la implementación de una etapa de detección de objetos antes de la clasificación podría mejorar significativamente el rendimiento del modelo. Esto implica la identificación y delimitación de áreas de interés, como los individuos a identificar, utilizando técnicas de detección de objetos como *Bounding Boxes*. Una vez que se hayan delimitado estas áreas, se pueden recortar y utilizar como entradas para los modelos de clasificación. Esto permitiría al modelo centrarse en las regiones de interés y reducir la influencia de elementos no relevantes en la imagen.

A medida que la tecnología de *deep learning* continúa avanzando, se pueden explorar arquitecturas de redes neuronales más avanzadas. Además de las redes MobileNetV2 y la *Arquitectura de Crohn* utilizadas en este estudio, se pueden probar arquitecturas más recientes y potentes. Por ejemplo, las redes con atención (*attention-based networks*) han demostrado ser efectivas en tareas de visión por computador. Investigar cómo estas arquitecturas pueden aplicarse a la clasificación de especies en imágenes de fototrampeo podría ser una dirección prometedora.

El uso de técnicas de *data augmentation* es esencial para mejorar la capacidad de generalización de los modelos. En futuras investigaciones, se puede profundizar en el refinamiento de estas técnicas. Esto podría incluir la exploración de técnicas más avanzadas de aumento de datos, como la generación de imágenes sintéticas o la aplicación de transformaciones específicas para mejorar la variabilidad de los datos.

11. Anexo I: Códigos

11.1. Código para hacer el Data Augmentation

```
# Preprocesamiento con Data Augmentation
data_augmentation = keras.Sequential(
    name="data_augmentation",
    layers=[
        layers.Rescaling(1.0/255),
        layers.Resizing(TARGET_IMG_HEIGHT, TARGET_IMG_WIDTH),

        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomRotation(0.1),

        tf.keras.layers.RandomContrast(factor = 0.5),
        tf.keras.layers.RandomBrightness(factor = 0.3, value_range=(0,1)),
        tf.keras.layers.GaussianNoise(0.1),

        tf.keras.layers.RandomTranslation(
            height_factor = 1,
            width_factor= 1,
            fill_mode='wrap',
            interpolation='bilinear',
            seed=None,
            fill_value=0.0,
        )
    ]
)
```

Figura 62: Código para hacer el Data Augmentation

11.2. Código para cargar el modelo *MobileNetV2*

```
modelo_base = tf.keras.applications.MobileNetV2(
    weights=INITIAL_WEIGHTS,
    include_top=False,
    input_shape=TARGET_SHAPE)

# Capas del modelo
modelo_base.trainable=False
```

Figura 63: Código para cargar el modelo *MobileNetV2*

11.3. Código para agregar las capas superiores al modelo *MobileNetV2*

```
# Capas Superiores
x = modelo_base.output
x = GlobalAveragePooling2D()(x)
x = Dropout(rate=DROPOUT_FACTOR)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(rate=DROPOUT_FACTOR)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(rate=DROPOUT_FACTOR)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(rate=DROPOUT_FACTOR)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(rate=DROPOUT_FACTOR)(x)
x = Dense(32, activation='relu')(x)
predictions = Dense(NUM_CLASSES, activation='softmax')(x)
```

Figura 64: Código para agregar las capas superiores al modelo *MobileNetV2*

11.4. Código de los bloques de capas de Crohn

```
def conv_block(inputs, filters):
    x = tf.keras.layers.Conv2D(filters, kernel_size=(3, 3), strides=1, padding='same')(inputs)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(DROPOUT_FACTOR)(x)
    return x

def conv_blocks(inputs, n_filters):
    x = conv_block(inputs, n_filters)
    x = conv_block(x, n_filters)
    x = conv_block(x, n_filters)
    return x
```

Figura 65: Código de los bloques de capas de Crohn

11.5. Código para crear el modelo Crohn

```
# Añadir capas
inputs = tf.keras.Input(shape=TARGET_SHAPE)

# Feature extraction blocks
x = conv_blocks(inputs, 32)
x = tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=2, padding='same')(x)
x = tf.keras.layers.Dropout(DROPOUT_FACTOR)(x)

x = conv_blocks(x, 48)
x = tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=2, padding='same')(x)
x = tf.keras.layers.Dropout(DROPOUT_FACTOR)(x)

x = conv_blocks(x, 56)
x = tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=2, padding='same')(x)
x = tf.keras.layers.Dropout(DROPOUT_FACTOR)(x)

x = conv_blocks(x, 64)
x = tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=2, padding='same')(x)
x = tf.keras.layers.Dropout(DROPOUT_FACTOR)(x)

x = conv_blocks(x, 64)
x = tf.keras.layers.MaxPooling2D(pool_size=(3, 3), strides=2, padding='same')(x)
x = tf.keras.layers.Dropout(DROPOUT_FACTOR)(x)

x = conv_blocks(x, 96)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(DROPOUT_FACTOR)(x)

# Classification layer
predictions = tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')(x)
```

Figura 66: Código para crear el modelo Crohn

11.6. Hiperparámetros para el modelo MobileNetV2

```
# Preprocesamiento
BATCH_SIZE = 32

TARGET_IMG_WIDTH = 224
TARGET_IMG_HEIGHT = 224
TARGET_IMG_CHANNELS = 3
TARGET_SIZE = [TARGET_IMG_WIDTH, TARGET_IMG_HEIGHT]
TARGET_SHAPE = [TARGET_IMG_WIDTH, TARGET_IMG_HEIGHT, TARGET_IMG_CHANNELS]

NUM_EPOCHS = 50

# Modelo
MODEL_NAME = "MobileNetV2"

CHECKPOINT_MONITOR = 'val_accuracy'
CHECKPOINT_MODE = 'max'

EARLYSTOP_MONITOR = 'val_loss'
EARLYSTOP_PATIENCE = 15

LR_MONITOR = 'val_loss'
LR_FACTOR = 0.5
LR_PATIENCE = 5
LR_MINLR = 0.0001

NUM_CLASSES = 8

hidden_size_1 = 1024
hidden_size_2 = 256

INITIAL_WEIGHTS = 'imagenet'


# Train
OPTIMIZER = "adam"
LOSS = 'sparse_categorical_crossentropy'
METRICS = ["accuracy"]

DROPOUT_FACTOR = 0
```

Figura 67: Hiperparámetros para el modelo MobileNetV2

11.7. Hiperparámetros para el modelo Crohn

```
# Preprocesamiento
BATCH_SIZE = 32

TARGET_IMG_WIDTH = 224
TARGET_IMG_HEIGHT = 224
TARGET_IMG_CHANNELS = 3
TARGET_SIZE = [TARGET_IMG_WIDTH, TARGET_IMG_HEIGHT]
TARGET_SHAPE = [TARGET_IMG_WIDTH, TARGET_IMG_HEIGHT, TARGET_IMG_CHANNELS]

NUM_EPOCHS = 100

# Modelo
MODEL_NAME = "Crohn"

CHECKPOINT_MONITOR = 'val_accuracy'
CHECKPOINT_MODE = 'max'

EARLYSTOP_MONITOR = 'val_loss'
EARLYSTOP_PATIENCE = 25

LR_MONITOR = 'val_loss'
LR_FACTOR = 0.5
LR_PATIENCE = 10
LR_MINLR = 0.0001

NUM_CLASSES = 8

INITIAL_WEIGHTS = 'imagenet'

# Train
OPTIMIZER = "adam"
LOSS = 'sparse_categorical_crossentropy'
METRICS = ["accuracy"]
```

Figura 68: Hiperparámetros para el modelo Crohn

Referencias

[1] S. Russell and P. Norvig, "Artificial intelligence: A modern approach, harlow (uk) &c," 2022.

[2] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] B. G. Weinstein, "A computer vision for animal ecology," *Journal of Animal Ecology*, vol. 87, no. 3, pp. 533–545, 2018.

[5] J. Krohn, G. Beyleveld, and A. Bassens, *Deep Learning Illustrated*. Addison-Wesley Professional, 2019.

[6] X. Su, X. Yan, and C.-L. Tsai, "Linear regression," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 3, pp. 275–294, 2012.

[7] C. Kingsford and S. L. Salzberg, "What are decision trees?," *Nature biotechnology*, vol. 26, no. 9, pp. 1011–1013, 2008.

[8] O. Kramer and O. Kramer, "K-nearest neighbors," *Dimensionality reduction with unsupervised nearest neighbors*, pp. 13–23, 2013.

[9] W. S. Noble, "What is a support vector machine?," *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[11] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[12] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[13] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[14] C. Boeree, "Psychology: the beginnings," *Retrieved April*, vol. 26, p. 2008, 2000.

[15] H. Wang and B. Raj, "On the origin of deep learning," *arXiv preprint arXiv:1702.07800*, 2017.

[16] A. M. Turing *et al.*, "On computable numbers, with an application to the entscheidungsproblem," *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.

[17] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and adaptive systems: fundamentals through simulations with CD-ROM*. John Wiley & Sons, Inc., 1999.

[18] A. M. Turing, *Computing machinery and intelligence*. Springer, 2009.

[19] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955," *AI magazine*, vol. 27, no. 4, pp. 12–12, 2006.

[20] B. Widrow and M. E. Hoff, "Adaptive switching circuits," tech. rep., Stanford Univ Ca Stanford Electronics Labs, 1960.

[21] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, vol. 479, p. 480, 1969.

[22] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

[23] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and Cooperation in Neural Nets: Proceedings of the US-Japan Joint Seminar held at Kyoto, Japan February 15–19, 1982*, pp. 267–285, Springer, 1982.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[25] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[27] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "Coca: Contrastive captioners are image-text foundation models," *arXiv preprint arXiv:2205.01917*, 2022.

[28] A. Salameh, "Artificial intelligence as a commons-opportunities and challenges for society," 2017.

[29] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[30] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[32] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. ."O'Reilly Media, Inc.", 2022.

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[34] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[35] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[36] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, PMLR, 2013.

[37] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[38] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[39] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[40] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[41] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pp. 818–833, Springer, 2014.

[42] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 111–118, 2010.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[48] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[49] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[50] D. Marin-Santos, J. A. Contreras-Fernandez, I. Perez-Borrero, H. Pallares-Manrique, and M. E. Gegundez-Arias, "Automatic detection of crohn disease in wireless capsule endoscopic images using a deep convolutional neural network," *Applied Intelligence*, vol. 53, no. 10, pp. 12632–12646, 2023.

[51] P. Meek, G. Ballard, A. Claridge, R. Kays, K. Moseby, T. O'brien, A. O'connell, J. Sanderson, D. Swann, M. Tobler, *et al.*, "Recommended guiding principles for reporting on camera trapping research," *Biodiversity and conservation*, vol. 23, pp. 2321–2343, 2014.

[52] A. F. O'Connell, J. D. Nichols, and K. U. Karanth, *Camera traps in animal ecology: methods and analyses*, vol. 271. Springer, 2011.

[53] M. S. Norouzzadeh, A. Nguyen, M. Kosmala, A. Swanson, M. S. Palmer, C. Packer, and J. Clune, "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 25, pp. E5716–E5725, 2018.

[54] J. Marcus Rowcliffe, C. Carbone, P. A. Jansen, R. Kays, and B. Kranstauber, "Quantifying the sensitivity of camera traps: an adapted distance sampling approach," *Methods in Ecology and Evolution*, vol. 2, no. 5, pp. 464–476, 2011.

[55] A. Swanson, M. Kosmala, C. Lintott, R. Simpson, A. Smith, and C. Packer, "Snapshot serengeti, high-frequency annotated camera trap images of 40 mammalian species in an african savanna," *Scientific data*, vol. 2, no. 1, pp. 1–14, 2015.

[56] S. Beery, G. Van Horn, and P. Perona, "Recognition in terra incognita," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 456–473, 2018.

[57] Aicensus, "Aicensus." Online; Accessed on: 2023.

[58] The Zooniverse Team, "Iberian camera trap project zooniverse." Online; Accessed on: 2023.

[59] Kaggle Inc., "Kaggle." https://www.kaggle.com/. Accedido en: Julio 2023.

[60] M. ISRAK AHMED, "Vegetable image dataset." https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset/. Accedido en: Julio 2023.

[61] "Vegetable image dataset." https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset. Accessed: 10 Julio 2023.